UNIVERSITAT DE BARCELONA

Treball final de grau

DOBLE GRAU DE MATEMÀTIQUES I ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Neutral Face Generator using Variational Autoencoders

Autor: Jorge Acosta Hernández

Director: Dr. Meysam Madadi
Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, September 13, 2020

Contents

Chapter 1

Introduction

1.1 Problem of study

Nowadays lack of data is an important problem for researchers. Quality datasets are difficult to gather. To solve the lack of data, researchers have two combinable approaches, either generate more data or develop better discriminative feature extraction algorithms that can be able to understand the important features of the data with smaller datasets. The objective of this project is to provide a tool to generate new data. In concrete we want to check if we are able to generate a concrete facial expresion from some images of a subject. We will center our project in generating neutral faces. This is the beginning, later this can be expanded to generate other kind of expresions. This can help to create labeled faces for research purposes.

The aim of this project is to develop a model which is able to generate a neutral face from a sample of images of the same person with different expressions. This expressions can be gathered from a video of that person just talking frontally to the camera. For doing this we will need a model which as it is able to learn meaningful features of the frontal faces. Also this model should be capable of reconstructing faces from this learned features. Gathering enough data for our model is another important point. Summing up, project challenges are :

- Finding a suitable model
- Training the model efficiently
- Gathering training data

1.2 Our approach

1.2.1 Finding a suitable model

Classical machine learning techniques have been recently overcome by Deep Neural Networks [1], it's true that in some specific problems they can beat DNN's, but in general DNN's are performing better in different types of problems and datasets. In order to learn features and generate data, our first idea was to use some kind of DNN. There are 2 principal models, Generative Adversarial Networks (GAN's) and Variational Autoencoders (VAE's). Recent papers[2] have shown that VAE's can learn better than GAN's data distribution. GAN structure is more complex to implement, as it requires the implementation of a generator and a discriminator.

Gathering many subject's frontal faces can be a hard job. VAE structure allows us to develop the project in 2 stages. Firstly, we are going to train a neutral face generator(as gathering only neutral faces is easier). Secondly, we are going to train an encoder which learns meaningful features from different subject's frontal faces using a smaller dataset. All in all we will be applying transfer learning to reduce the effort in gathering data taking, advantage of the encoder-decoder VAE's structure.

1.2.2 Training the model efficiently

For develop the project we had 2 options, PyTorch or TensorFlow. Both machine learning libraries could fit in with our purpose, but TensorFlow has some advantages :

- A better support for CUDA
- A larger community
- Higher functionalities, as flipping tensors along with dimension, checking tensors for infinity or NaN or providing suppport for faster Fourier transforms that traduces in less computation time.

1.2.3 Gathering training data

Data is the second most important part of any DNN project (the choosed model is the first one). For our project we needed a tool that allowed us to :

- Crop faces from images
- Distinguish frontal faces
- Distinguish neutral frontal faces

The selected tool was OpenFace, a open-source project. OpenFace showed to perform very well in face recognition providing helpful data to achieve our objectives. OpenFace is easy to use and their developers are always up to solve any doubt.

Chapter 2

State of the art

2.1 Image generation

Generative models is a recent powerful way of learning any kind of data distribution using unsupervised learning. This model's aim is to learn the true distribution of a dataset. Once the distribution is learned the model is able to generate new data points with some variations. There are two principal approaches Generative adversarial networks (GAN) and Variational Autoencoders (VAE). VAE aims at maximazing the lower bound of the data log-likelihood and GAN aims to achieve an equilibrium between a generator and a discriminator.

Reasearchers of DeepMind have introduced Vector Quantised-Variational AutoEncoder (VQ-VAE) [2], based on Variationl AutoEncoders (VAE) for large scale image generation. This new model can compete with state-of-the-art generative model BigGAN [3] in synthesizing high-resolution images while delivering broader diversity and overcoming some native shortcomings of GANs as lack of diversity (models unable to capture the diversity of the true distribution) or mode collapse problems (generator produces limited varieties of samples).

These issues prompted DeepMind to explore the use of Variational AutoEncoders (VAE), an unsupervised learning approach that trains the model to learn representations from datasets. In their NIPS 2017 paper Neural Discrete Representation Learning, DeepMind researchers introduced VQ-VAE, or Vector Quantised Variational AutoEncoder, a VAE variant that comprises an encoder that transforms image data into discrete rather than continuous latent variables (representations), and a decoder which reconstructs images from these variables. DeepMind reasearchers show how VQ-VAE can compress images into a latent space which is about 50 times smaller for ImageNet and 200 smaller for FFHQ Faces. This makes the latent space more tractable and compact [2].

Researchers used ImageNet and FFHQ as the datasets in their experiments. Trained on ImageNet 256 x 256 images, VQ-VAE generated comparable high-fidelity images and delivered higher diversity than BigGAN. On FFHQ 1024 x 1024 high-resolution face data, VQ-VAE generated realistic facial images while still covering some features represented only

sparsely in the training dataset. The paper also discussed other evaluation metrics to test VQ-VAE performance.

2.2 Face generation

Nowadays face generation is a domain in constant change. Thanks to Deep learning results are getting better quickly. The new state-of-the-art model is called StyleGAN2. Researchers led by Tero Karras published a paper[4] where they analyze the capabilities of the original StyleGAN model and propose some improvements. StyleGAN was developed by Nvidia reseachers and its generation method is based on GAN's. This generative model was the first one able to generate high quality images and also allow to control the style of the generated image.

Researchers have identified and fixed several image quality issues in StyleGAN, improving the quality further and considerably advancing the state of the art in several datasets. They have also improve the training performance. This model is able to train quicker than StyleGAN.

Despite this improvements the paper mention that it is still a big challenge to reduce the data requirements as gathering quality data continues to be a big problem.

Chapter 3

Deep learning background

In this section we are going to make a brief review over the history background of Deep Learing, and also take a look to some basic concepts.

3.1 Brief History

Deep Learning has been a hottopic the last years, improving the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains. Deep Learning has a very big impact in the way we understand the world nowadays, it has applications in important fields as health informatics [5], autonomous driving [6] or natural language processing [7].

In 1949 Donald Hebb introduced the Hebbian Learning Rule in his book[8], that made him to be known as the father of Neural Networks, with claims like : "Cells that fire together, wire together" emphasizing the fact that the connection between two units should be strengthened as the frequency of co-occurrences of these two units increase. This statement can be viewed using modern machine learning notation as :

$$\Delta w_i = \eta x_i y \tag{3.1}$$

where Δw_i stands for the change of the synaptic weights (w_i) of neuron i, of which the input signal is x_i, y denotes the post synaptic response and η is a learning rate.

The next steps were done in 1970 by Seppo Linnainmaa, who used an initial version of the Backpropagation algorithm as a tool for estimating the effects of arithmetic rounding errors on the result of complex expressions [9][10]. Further on we can find more important contributions for the mordern version of this algorithm made in 1974 by Paul Werbos[11] and in 1986 by David Rumelhart [12].

In 1980 Kunihiko Fukushima introduced Neocogitron[13], which inspired Convolutional Neural Networks (CNN's) that will be explained in next sections. Also in this decade two important advances took place, in first place in 1982 John Hopfield introduced the so-called

Hopfield Network [14], this networks were made to recognise patterns and store them using binary treshold nodes. Hopfield Networks consists of several binary neurons connected all between them with an associated weight w_{ij} to the connection, so they can be described as an undirected graph $G = \langle V, f \rangle$ where V is a set of binary neurons and $f : V^2 \to \mathbb{R}$ is a function linking pairs of units to a real value, the weight w_{ij}. Hopfield Networks usually don't have autoconnections $w_{ii} = 0$ in the neurons and the weights are symmetric $w_ij = w_ji$, the update of the units follows the next rule :

$$s_i \leftarrow \begin{cases} +1 & \text{if } \sum_j w_{ij} s_j \geq \theta_i, \\ -1 & \text{otherwise.} \end{cases} \tag{3.2}$$

where s_i and θ_i are the state and the threshold of unit i. The objective of the network is to minimize the energy function

$$E = -\frac{1}{2} \sum_{i,j} w_{ij} s_i s_j + \sum_i \theta_i s_i \tag{3.3}$$

it deacreses or stays the same while the network updates. That is why the network will always converge. The problem is that this convergence can finish in a local minimun.

The second important event in the eighties was the development of the Harmonium or more commonly known as Restricted Boltzmann Machine introduced by Paul Smolensky [15] based on the publication one year before of the Boltzmann Machine by Hilton, Ackley and Sejnowski[16]. The main difference between these two networks is that the Restricted Boltzmann machine neurons form a bipartitte graph. Which has applications in dimensionality reduction[17], classification[18], feature learing[19], topic modelling[20] and more fields. They can be trained either supervised or unsupervised ways, depending on the problem to solve. The restriction over the distribution of neurons allows to apply the gradient-based contrastive algorithm [21]. This networks are formed by binary-valued hidden and visible units, with a matrix of weights $W = (w_{i,j})$ representing the connection between a hidden unit h_j and a visible unit v_i. Both visible and hidden units have a bias weight a_i and b_i respectively. As hopfield networks they have a defined energy function, where (v, h) are a pair of boolean vectors

$$E(v, h) = -a^{\mathrm{T}} v - b^{\mathrm{T}} h - v^{\mathrm{T}} W h \tag{3.4}$$

we can see that the structure starts to look like modern Neural Networks (NN).

One of the most important publications was made by Yann LeCunn published in 1990. LeCunn's article shows the practical development of NN in real world problems[22] appliying the backpropagation algorithm and showing for example a NN that learned handwritten digits, showing it's posterior performance with an error rate of only 1%, LeCunn and his collegues continued with the research, and in 1998 showed amazing results outperforming the most part of classic machine learing methods in several real world problems[23].

In 1991 Sepp Hochreiter identifies the problem of vanishing gradient which can make the learning of deep neural network extremely slow and almost impractical. This was not a fundamental problem for all neural networks, just the ones with gradient-based learning

methods. The source of the problem turned out to be certain activation functions. A number of activation functions condensed their input, in turn reducing the output range in a somewhat chaotic fashion. This produced large areas of input mapped over an extremely small range. In these areas of input, a large change will be reduced to a small change in the output, resulting in a vanishing gradient. Two solutions used to solve this problem were layer-by-layer pre-training and the development of long short-term memory.

In 2006 Geoffrey Hinton, Ruslan Salakhutdinov, Osindero and Teh published a paper[24] in which they stacked multiple RBMs together in layers and called them Deep Belief Networks. The training process is much more efficient for large amount of data.

Then the GPU revolution started, Ranja, Madhavan and Andrew Ng published a paper [] in 2009 advocating for the use of GPUs for training Deep Neural Networks to speed up the training time by many folds. This could bring practicality in the field of Deep Learning for training on huge volume of data efficiently. Also in 2009 ImageNet[25], a huge database of 14 million labeled images, was launched by the professor Fei-Fei Li and his team of the Standford University. This dataset is used nowadays as a benchmark for deep neural networks.

The vanishing gradient problem was mostly solved in 2011 by Yoshua Bengio, Antoine Bordes, Xavier Glorot they showed[26] how ReLU activation function can avoid it. Next year, 2012, AlexNet, a GPU implemented CNN model designed by Alex Krizhevsky, wins Imagenet's image classification contest with accuracy of 84%. It is a huge jump over 75% accuracy that earlier models had achieved. This win triggers a new deep learning boom globally.

Related with this project we can highlight Ian Goodfellow work [27], developing GAN, this new type of networks allow to generate new data in such different fields as fashion, art or science. GAN's and VAE's could be the key of creating a whole new set of applications, as they seem to learn distribution of datasets, simulating what we can call human imagination.

3.2 Basic notions

We will make a brief introduction to some basic concepts to understand the motivation and the basic structures that form neural networks

3.2.1 Biological inspiration

Neural Networks inspiration consists on emulate the biological model or at least what we know about it. Something interesting is that we are also using them to try to understand how the real neurons work. They could be a very useful tool to check the beahviour of our brain.

Our brain follows approximately the model showed in Figure 3.1[1]. Basically the dendrites will play the role of the inputs, the soma acts as the summation and posterior activation

[1]Image extracted fomr www.bla.com

function, the axon is the output of this neuron and when it connects to other neuron by the dendrites the space between them called synapse acts has the weight of the input to the next neuron. This will be the equivalence between biological components and mathematical objects.

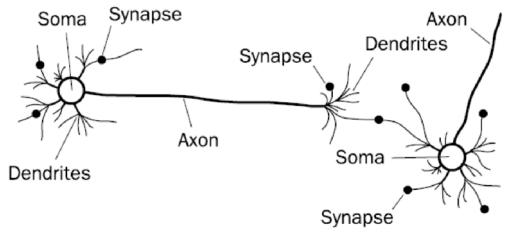


Figure 3.1: Neuron model

An easy way of getting the inspiration can be thinking of how the visual cortex works, the information goes into the retina (input layer) and is passed to a region called LGN (pre-processing) and then is passed to a region called V1 (first hidden layer) which extracts simple visual forms as edges or corners, this region triggers another near region called V2 (second hidden layer) and another region called V4 that is sensitive to intermediate visual forms, feature groups, etc, and then PIT and AIT are also triggered that perform recognition of high level objects. So each region is specialized in a type of recognition and all of them are related, this is the behavior that we want to emulate. Figure 3.2 illustrates this beahviour.

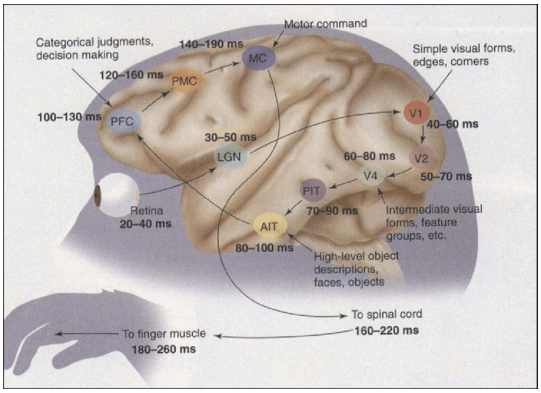


Figure 3.2: Visual Processing of Object Structure. Image extracted from Simon J. Thorpe paper [28]

3.2.2 Perceptron

The basic unit of the Neural Networks is the perceptron, it can be easily described mathematically as some inputs $x_1, ..., x_n$, some weights $w_1, ..., w_n$ each of them associated to one of the inputs, a bias b, a pre-activation function $a(x)$ and an activation function $f(x)$. The computation of the output is made doing the next simple math :

- Pre-activation : $a(x) = \sum_i w_i x_i + b = b + w^T x$

- Output function : $output = f(a(x)) = f(\sum_i w_i x_i + b)$

We can use many activation functions but the idea is to have a function that acts as a threshold between what values should make the network totally activates giving an output of approximately 1 or otherwise very near to 0, typical activations functions are :

- Linear activation function : $f(x) = x$, this function doesn't performs any squashing of the input, so it is not lower or upper bounded. It is not very insteresting. Figure 3.3 shows its shape.

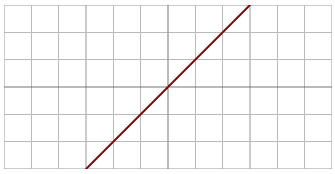


Figure 3.3: Linear activation function

- Sigmoid activation function : $f(x) = sigm(x) = \frac{1}{1+e^{-x}}$, this function squashes the neuron pre-activation function between 0 and 1, so its is upper and lower bounded. It is always positive and strictly increasing as we can see in Figure 3.4.

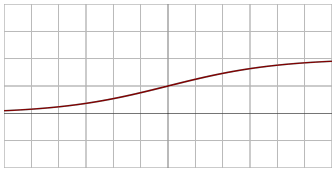


Figure 3.4: Sigmoid activation function

- Hyperbolic tangent activation function : $f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$, this function squashes the neuron pre-activation function between -1 and 1, so it is upper and lower bounded and it can be positive or negative as we can see in Figure 3.5.

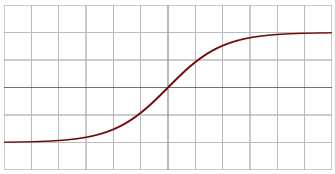


Figure 3.5: Hyperbolic tangent activation function

- Rectified linear activation function : $f(x) = ReLU(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$, this function is only lower bounded by 0, so it is always non-negative. It is strictly increasing. And in practice tends to give neurons with sparse activities, this means that it tends to give some neurons which value is exactly 0 as we can see in Figure 3.6.

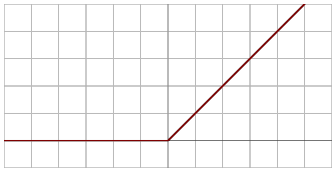


Figure 3.6: Rectified linear activation function

- Exponential linear activation function : $f(x, \alpha) = ELU(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$, this function is only lower bounded by $-\alpha$. It is strictly increasing. It doesn't have 0 slope in any point as we can see in Figure 3.7.

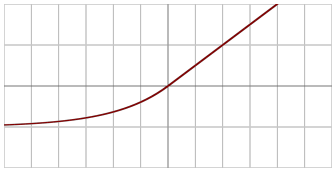


Figure 3.7: Exponential linear activation function

Perceptron is a linear classifier, so if the training set is linearly separable it will find a suitable hyperplane to solve the problem. If the problem is not linearly separable then it will fail. As the perceptron guarantees the convergence to some solution in linearly separable sets it can find different solutions. There is an special case called *the perceptron of optimal stability* which is equivalent to the support-vector machines (SVM).

3.2.3 Multilayer neural network

The main problem of the perceptron is that it doesn't work on nonlinearly separable sets, and the most part of classification sets are not of this type. The next step consists on connecting several perceptrons between them in a layer structure, in general it has in at least 3 layers, a input layer, a hidden layer and an ouput layer.

We can describe a single hidden layer neural network mathematically as a set of inputs $x_1, ..., x_n$, some weights $w_{11}, ..., w_{nm}$ representing the connection from neuron i to neuron j of the next layer, it is easier to describe it as a matrix for each layer W^1 for the hidden layer and W^2 for the output layer, matix form is always cleaner, a set of biases $b_{11}, ..., b_{nm}$ representing

the bias of neuron i in layer j or as vectors b^1 for the hidden layer and b^2 for the output layer. Figure 3.8 might be helpful to understand the structure. An activation function $f(X)$ which can be different in each layer or even perceptron, but for simplicity we will use the same for the whole net, so the output of the net follows the following computation :

- Pre-activation : $a(x)_i = \sum_j W^1_{ij} x_j + b_{i1} = b^1 + W^1 x$
- Hidden layer activation : $h(x) = f(a(x))$
- Output layer activation : $O(x) = g(b^2 + W^2 h^1(x))$, with $g(x)$ being the output activation function

The output function is usually a sigmoid for binary classification problems, but if we have more than 2 classes, then we need a way to perform multiclass classification, and we will like to estimate the conditional probability $p(y = c|x)$ of each of the classes. For this we use the softmax activation function in the output as shown in Equation 3.5.

$$O(x) = softmax(x) = \left[\frac{e(x_1)}{\sum_j e(x_j)}, ..., \frac{e(x_n)}{\sum_j e(x_j)} \right]^T \tag{3.5}$$

With n being the number of classes, softmax is strictly possible as it represents a probability and it sums to one. The predicted class will be the highest estimated probability.

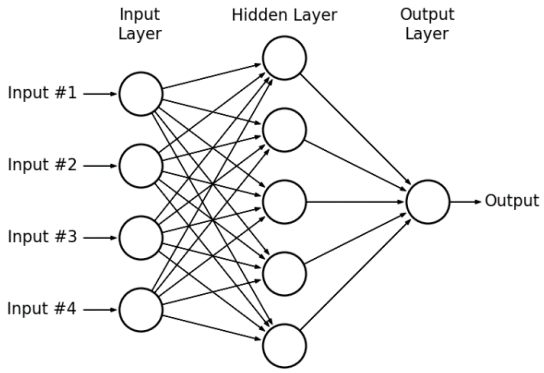


Figure 3.8: Multilayer perceptron. Image extracted from paper [29]

Multilayered hidden neural networks are the generalization of single hidden layer neural networks. Let say we have L hidden layers, $W^1, ...W^L$ weights, $b^1, ..., b^L$ biases, then in a matrix form we will have :

- Layer pre-activation, for $k > 0$, $(h^0(x) = x)$: $a^k(x) = b^k + W^k h^{k-1}(x)$
- Hidden layer activation (k from 1 to L) : $h^k(x) = f(a^k(x))$
- Output layer activation $(k = L + 1)$: $h^{(L+1)}(x) = O(a^{(L+1)}(x)) = f(x)$

A single hidden layer neural networks with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units are as the universal approximation theorem [30] states. This result also applies for sigmoid, tanh and many other hidden layer activation functions. This result doesn't mean there is a learning algorithm that can find the necessary parameter values. We will see how to solve this in the following section.

3.2.4 Backpropagation Algorithm

The backpropagation algorithm is the basis of the training used for feedforward neural networks for training. There are generalizations of the algorithm that can be also applied to other networks. In general and as an intuition, the algorithm computes a efficient way of changing the weights of a network considering the error made by the network in the last computation.

Really not necessary to have labeled data, what we need is to be able to compute a loss function that gives us an estimation of the error made by the network solving our problem. But it's easier to understand with a classification problem, and just comparing the predicted label with the ground truth.

The modern version of this algorithm was publisehd by Rumelhart[12] as we saw in previous section. As the derivation of this algorithm is a little bit complex we will show it step by step.

SGD and Loss function

The training problem becomes an optimization problem. Lets say θ is the set of all the parameters in the connection matrices, x^t is the input of the network for the sample number t and y^t is the associated label that we want to predict, then $f(x;\theta)$ is the function that gives us the output of the network. Then we have a loss function $l(x,y)$ that compares the output with the label and a regularization term $\omega(\theta)$ that penalizes ceratin values of θ, λ is just an hyperparameter to decide how much penalization we want to apply. The optimization is performed as :

$$argmin_\theta \frac{1}{T}\sum_t l(f((x^t;\theta),y^t) + \lambda\omega(\theta) \tag{3.6}$$

This is the general framework for minimizing the loss function, as this is an optimization problem the stochastic gradient descent (SGD) algorithm fits well. The algorithm updates the weights after each example in the following way :

- Initialize randomly the weights $\theta = W^1, b^1, ..., W^{L+1}, b^{L+1}$
- For N iterations and for each training example (x^t, y^t) :
    1. $\Delta = -\nabla_\theta l(f(x^t,\theta),y^t) - \lambda\nabla_\theta\omega(\theta)$ this is the direction where we will get the biggest decrease in the loss function.

2. $\theta \leftarrow \theta + \alpha\Delta$

Being α the learning rate. To apply this algorithm we need the loss function $l(f(x^t, \theta), y^t)$, some procedure to compute the gradient $\nabla_\theta l(f(x^t, \theta), y^t)$, the regularizer form $\lambda\nabla_\theta\omega(\theta)$, and a method to initialize the weights and biases.

The loss function is a key item, it can be different depending on our problem, for typical classification problems is usual to use cross-entropy, being $f(x)c$ the probability that the input x belongs to the class c,

$$l(f(x), y) = -\sum_c 1_{(y=c)} log(f(x)_c) = -log(f(x)_y) \tag{3.7}$$

Output layer gradient

This is the easy part, first of all we want to compute the partial derivatives of each output neuron and then combine them. So we compute the partial derivative of the networks output with respect to the c component of the output vector,

$$\frac{\partial}{\partial f(x)_c} - log(f(x)_y) = \frac{-1_{(y=c)}}{f(x)_y} \tag{3.8}$$

if c is not equal to y then the quantity can be a constant, so that is why we use the indicator $1_{(y=c)}$, to obtain the gradient we need to put all togheter in vector form getting,

$$\nabla_{f(x)} - log(f(x)_y) = \frac{-1}{f(x)_y}[1_{(y=0)}, ..., 1_{(y=C-1)}] = \frac{-e(y)}{f(x)_y} \tag{3.9}$$

where $e(y)$ represents a vector with all zeros and a 1 in some of its posistions. Now we need to compute the gradient for the pre-activation function , first let see the partial derivative of the loss function with respect to the c element, this derivation is pretty long involving some mathematical tricks and really big fractions and summations for simplicity we will only give the result :

$$\frac{\partial}{\partial a^{(L+1)}(x)_c} - log(f(x)_y) = -(1_{(y=c)} - f(x)_c) \tag{3.10}$$

$$\nabla_{a^{(L+1)}(x)} - log(f(x)_y) = -(e(y) - f(x)) \tag{3.11}$$

Hidden layer gradient

Now things get a little bit more complex, now we will show the computation in a more general way, as looking neuron by neuron makes things really difficult to understand. First of all we will use the chain rule to get a generalized formulation of the gradient with with respect to any hidden layer.

From the chain rule we know that if we have a function $p(a)$ that can be written as a function of intermediate results $q_i(a)$ then we have,

$$\frac{\partial p(a)}{\partial a} = \sum_i \frac{\partial p(a)}{\partial q_i(a)} \frac{\partial q_i(a)}{\partial a} \tag{3.12}$$

then we set a as the activation of a neuron in some layer, $q_i(a)$ the pre-activation in the layer above and $p(a)$ as the loss function. Knowing this lets see the computation of the partial derivative in each hidden layer,

$$\frac{\partial}{\partial h^k(x)_j} - log f(x)_y = \sum_i \frac{\partial - log(f(x)_y)}{\partial a^{(k+1)}(x)_i} \frac{\partial a^{(k+1)}(x)_i}{\partial h^k(x)_j} = \frac{\partial - log(f(x)_y)}{\partial a^{(k+1)}(x)_i} W_{ij}^{(k+1)} = (W_{j\cdot})^T (\nabla_{a^{(k+1)}(x)} - log(f(x)_y) \tag{3.13}$$

where in the second equality we have $a^{k+1}(x)_i = b_i^{k+1} + \sum_j W_{ij}^{k+1} h^{(k)}(x)_j$ and applying the partial derivative respect to $h^k(x)_j$ will give us the scalar W_{ij}. So the gradient is,

$$\nabla_{h^k(x)} - log(f(x)_y) = (W^{(k+1)})^T (\nabla_{a^{(k+1)}(x)} - log(f(x)_y) \tag{3.14}$$

Now we need to also know the form of the gradient respect to the pre-activation function, an important reminder is that $h^k(x)_j = g(a^k(x)_j)$. So the partial derivative will be,

$$\frac{\partial}{\partial a^k(x)_j} - log(f(x)_y) = \frac{\partial - log(f(x)_y)}{\partial h^k(x)_j} \frac{\partial h^k(x)_j}{\partial a^k(x)_j} = \frac{\partial - log(f(x)_y)}{\partial h^k(x)_j} g'(a^k(x)_j) \tag{3.15}$$

and the gradient will be,

$$\nabla_{a^k(x)} - log(f(x)_y) = (\nabla_{h^k(x)} - log(f(x)_y))^T \nabla_{a^k(x)} h^k(x)) = (\nabla_{h^k(x)} - log(f(x)_y)) \odot [..., g'(a^k(x)_j, ...] \tag{3.16}$$

Parameter gradient

Now we will derive the partial derivatives and gradients respect to the weights and biases. Knowing that $a^k(x)_i = b_i^k + \sum_j W_{ij}^k h^{(k-1)}(x)_j$ we can compute the partial derivatives respect to the weights :

$$\frac{\partial}{\partial W_{ij}^k} - log(f(x)_y) = \frac{\partial - log(f(x)_y)}{\partial a^k(x)_i} \frac{\partial a^k(x)_i}{\partial W_{ij}^k} = \frac{\partial - log(f(x)_y)}{\partial a^k(x)_i} h_j^{k-1}(x) \tag{3.17}$$

Then the gradient of the weights will have the form,

$$\nabla_{W^k(x)} - log(f(x)_y) = \left(\nabla_{a^k(x)} - log(f(x)_y)\right) h^{(k-1)}(x)^T \tag{3.18}$$

And for the biases we have (partial derivatives and gradient) :

$$\frac{\partial}{\partial b_i^k} - log(f(x)_y) = \frac{\partial - log(f(x)_y)}{\partial a^k(x)_i} \frac{\partial a^k(x)_i}{\partial b_i^k} = \frac{\partial - log(f(x)_y)}{\partial a^k(x)_i} \tag{3.19}$$

$$\nabla_{b^k(x)} - log(f(x)_y) = \nabla_{a^k(x)} - log(f(x)_y) \tag{3.20}$$

All together, backpropagation

Finally we can put all together to efficiently compute the parameter gradients, and be able to apply SGD. Summing up last computations we have (we assume a forward propagation has been made before) :

- output gradient : $\nabla_{a^{(L+1)}(x)} - log(f(x)_y) \leftarrow -(e(y) - f(x))$
- hidden layer parameter gradients : $\nabla_{W^k(x)} - log(f(x)_y) \leftarrow (\nabla_{a^k(x)} - log(f(x)_y))h^{(k-1)}(x)^T$ and $\nabla_{b^k(x)} - log(f(x)_y) \leftarrow \nabla_{a^k(x)} - log(f(x)_y)$
- hidden parameter gradients of below layer : $\nabla_{h^{k-1}(x)} - log(f(x)_y) \leftarrow (W^{(k)})^T(\nabla_{a^{(k)}(x)} - log(f(x)_y)$
- hidden parameter gradients of below layer before activation : $\nabla_{h^{k-1}(x)} - log(f(x)_y) \leftarrow (\nabla_{h^{k-1}(x)} - log(f(x)_y)) \odot [..., g'(a^{(k-1)}(x)_j, ...]$

We can see that except for the output gradient the expressions on the left depend on other parts of the neural network. So the idea is to apply the formulas in order so we have all the right side is always computed.

3.3 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN's) also known as Space Invariant Artificial Neural Networks (SIANN) are an special type of networks that are mostly applied on images. The main difference of this structure with a multilayered hidden neural network is basically that a CNN expects a numerical matrix as input. This type of network is very useful to classify images, cluster images by similarity or perform object recognition.

CNN's were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

A CNN expects an input 4-D tensor with the shape of (numImages, imageHeight, imageWidth, imageDepth), then the output will be also a 4-D tensor (numImages, newimageHeight, newimageWidth, newimageDepth).

A Convolutional layer consists of a bunch of kernels, these kernels are applied to the image giving us a new depth and also a new image. The size of the image can be reduced or not depending on how the kernel is applied, there are 2 variables that control this, stride and padding. Padding controls how many zeros are added to the border of the image and the stride controls the number of steps that the kernel moves forward in each kernel application. Figure 3.9 shows how a called padding of "SAME" maintain the size of the image adding just one line of zeros to the border of the image.

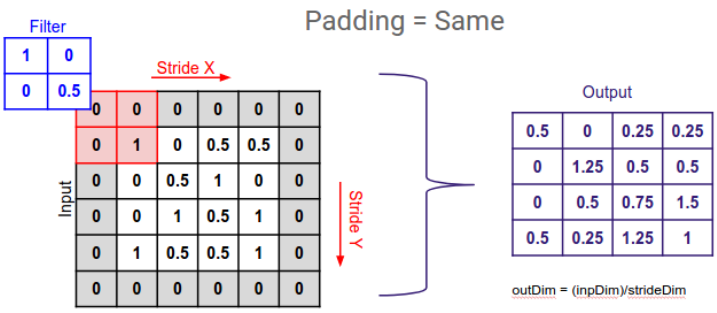


Figure 3.9: Padding example. Image extracted from article : Deep Learning: Convolutional Neural Networks from mc.ai

Each cell of the kernel represents a weight and this weights are adjusted depending on the loss function of the network, the structure include a bias. So many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting.

Another important feature of this networks that can be used or not is pooling, it's aim is to decrease the computational power required to process the data through dimensionality reduction. Pooling can also be useful to extract dominant features that are rotational and positional invariant of the images. There are two types of pooling, Max Pooling and Average Pooling. Max Pooling just returns the maximm value from the portion of the image covered by the window. Average Pooling returns the average of all the values covered by the window. In terms of performance Max Pooling gives in general better results than Average Pooling, despite the fact that both of them can be very useful. Figure 3.10 illustrates a combination of convolutional layers and max-pooling to extract features from an image.

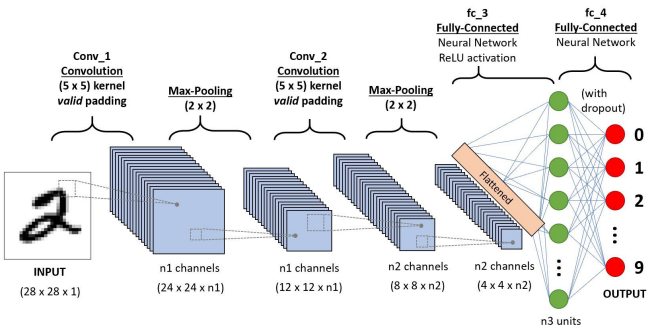


Figure 3.10: CNN applied for feature extraction. Image extracted from article : A Comprehensive Guide to Convolutional Neural Networks, by Sumit Saha

3.4 Dropout

Overfitting is an important issue in neural networks, we always want to avoid it and is easy that a network get overfitted. As we are exploring a space of complex functions where there might be a lot of local minimums, as deep neural networks are composed of lots of parameters we can be in a high variance and low bias situation.

The idea of dropout is to cripple the neural network by removing hidden units stochastically. We select a rate and then set hidden units to 0 with a probability equal to the selected rate, what we are looking for is that the hidden units get more general ad they cannot co-adapt to the other units. The usual way to implement dropout is using a random binary mask m^k for each hidden layer.

$$h^k(x) = f(a^k(x)) \odot m^k \tag{3.21}$$

Then the computations of the gradients change a little bit :

$$\nabla_{a^k(x)} - log(f(x)_y) \leftarrow (\nabla_{h^k(x)} - log(f(x)_y)) \odot [..., g'(a^k(x)_j, ...] \odot m^k \tag{3.22}$$

Dropout has shown very goods results beating regular backpropagation on many datasets[31], it seems to be a technique that we should always incorporate on our neural networks or at least test it. Dropout allows our models to avoid overfitting and get a better generalization.

3.5 Batch Normalization

When a Deep Neural Network is training the distribution of each layer's input changes, as the parameters of the previous layers change. This is called internal covariate shift. Batch Normalization (BN)[32] consists on normalizing each training mini-batch. BN allows us to use higher learning rates and be less careful about initialization. It can act as a regularizer avoiding in some cases the use of Dropout.

BN doenst normalize inputs and outputs of each layer jointly. It normalizes each scalar feature indepently, by making it having the mean of zero and the variance of 1. So for a layer with inputs $x_1, ..., x_d$ the normalization will be performed as,

$$\hat{x}_k = \frac{x_k - E[x_k]}{\sqrt{Var[x_k]}} \tag{3.23}$$

were the expectation and variance are computed over the training data set.

Normalizing each input of a layer could change what a layer represents. To fix this BN makes sure thate the transformation inserted in the network can represent the identity transform. For this BN introduces for each activation x_k two parameters γ_k and β_k, which scale and shift the normalized value,

$$y_k = \gamma_k \hat{x}_k + \beta_k \tag{3.24}$$

these parameters are learned as the original model parameters, restoring the representation power of the network.

Consider a mini-batch $B = x_1, ..., x_m$, then BN is performed as follows,

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{3.25}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \tag{3.26}$$

$$\hat{x}_i \leftarrow \frac{\hat{x}_i - \mu_B}{\sqrt{\sigma_B^2 + \hat{x}_i + \epsilon}} \tag{3.27}$$

$$y_k = \gamma_k \hat{x}_k + \beta_k \tag{3.28}$$

Were parameter ϵ in Equation 3.27 is a constant added to the mini-batch for numerical stability. Parameters γ_k and β_k in Equation 3.28 have to be learnead and they are not exclusive of each training sample x_i, they involve the whole mini-batch.

Chapter 4

Variational Autoencoder (VAE)

4.1 What is an Autoencoder?

An Autoencoder is a special type of artificial neural network architecture with the purpose of learning how to codify in an efficient way a dataset. We could say that the network should learn a reduced representation of the dataset.

The idea behind autoencoder is pretty easy to understand, first of all our goal is that the input and the output of the network are the same or at least very similar. In order to achieve this the loss function just computes the difference between them, using different measures(1-norm , 2-norm, ...) depending on the problem.

Autoencoders are different from classification networks, as they don't need labeled data, thus can be trained unsupervisedly, we only use the inputs x^t in the training. The main uses of this kind of networks are :

- Automatically extract meaningful features from data, something similar to PCA. Autoencoders learn how to efficiently represent the data.
- Leverage the availability of unlabeled data
- Add a data-dependent regularizer to trainings
- Image processing

Autoencoders are quite simple, it is a feed-forward neural network trained to reproduce its input in the output layer. Autoencoders have 2 differnt parts, the first one takes the data and learns how to represent it in the most efficient way, this is called the encoder. Then the second part takes this reduced representation and learns how to reconstruct the data with the least errors possible, this is called the decoder. Figure **??** shows the basic structure of an autoencoder.

Some common practice is to maintain the same structure in both parts, even sometimes is used to tied weights[34], this means that if the weights of the encoder are W then the weights in the decoder will be W^T. So as we can see is not a very complex definition.

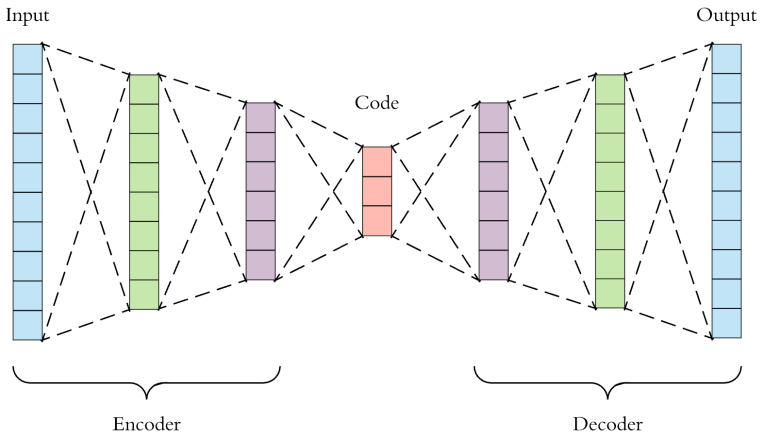


Figure 4.1: Autoencoder schema. Image extracted from article[33] written by Alkhayrat, Maha and Aljnidi, Mohamad and Aljoumaa, Kadan

We can describe it's structure mathematically very easy, being f the encoder, g the decoder, X our dataset and C our codified space, and l the choosen loss function.

- $f : X \to C$
- $g : C \to X$
- $loss = argmin_\theta \, l(X, (f \circ g)(X))$

The election of the loss function depends on the type of data that we have :

- For binary inputs the most common choice is the well known cross-entropy loss function : $l(x, f(x)) = -\sum_k (x_k log(f(x_k)) + (1 - x_k) log(1 - f(x_k))$
- For real-valued inputs the most common choice is the squared euclidean distance loss : $l(x, f(x)) = -\frac{1}{2} \sum_k (f(x_k) - x_k)^2$ and normally a linear activation function is used in the output layer

For both cases the gradient $\nabla_{a(x^t)} l(f(x^t))$ has a very simple form : $\nabla_{a(x^t)} l(f(x^t)) = f(x^t) - x^t$. Then parameter gradients are obtained by regular backpropagation as Autoencoders are feed-forward networks. Note that if we use tied weights, this gradient $\nabla_W l(f(x^t))$ is the sum of two gradients, as W is present in the encoder and the decoder.

4.2 Variational Autoencoder (VAE)

Variational Autoencoders [35] are a really smart modification of Autoencoders, they are part of the generative models, like Generative Adversarial Networks. Basically a VAE is a directed probabilistic grahical model (DPGM) whose posterior is approximated by a neural

network. The aim is to create a space which is distributed similar to a known distribution probability so we can sample from it and generate data this way.

So VAE's are based on variational inference, the basic notion of this area is Information concept described mathematically as $I = -log(p(x))$. A higher probability of and event x gives us less information and lower probability gives us more information. Other useful concept is Entropy, defined as $H = -\sum_i log(p(x_i))$

Other important concept is the Bayes Theorem, defined as the probability of a hypothsis, z, given some new data x, is denoted as $p(z|x)$ and is given by :

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{4.1}$$

Where $p(x)$ is the probability of the data x, $p(z|x)$ is the probability of the data given a hypothesis z, and $p(z)$ is the probability of that hypothesis z. Bayes Theorem arises directly out of the conditional probability axiom, which can be also derived directly of the definition from the joint probability.

As we want to approximate the latent space generated by the Autoencoder to a known distribution we need a way of measuring the difference between distributions. The mathematical object to describe this is the Kullback-Leibler divergence, also called realtive entropy.

$$D_{KL}(P||Q) = -\int p(x) log\left(\frac{q(x)}{p(x)}\right) = \int p(x) log\left(\frac{p(x)}{q(x)}\right) \tag{4.2}$$

Note that the Kullback-Leibler divergence is not symmetric :

$$D_{KL}(P||Q) \neq D_{KL}(Q||P) \tag{4.3}$$

On the right side we are taking the expectation of the information difference with respect to P distribution, while on the left side we are taking the expectation with respect to the Q distribution. Also it is remarkable that it's called a divergence and not a metric as metrics must be symmetric. An other important fact is that the Kullback-Leibler divergence is always non-negative.

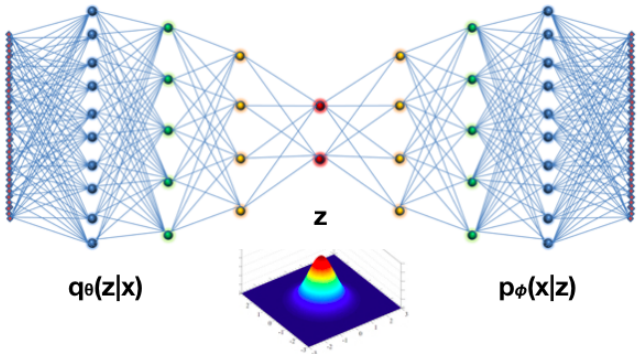


Figure 4.2: VAE schema. Image extracted from tutorial[36] written by Stephen G. Odaibo

Once we have this mathematical tools we can show how a VAE works and why [36]. The encoder portion of a VAE yields an approximate posterior distribution $q(z|x)$, and is parametrized on a neural network by weights collectively denoted a. Hence we more properly write the encoder as $q_\theta(z|x)$. Similarly, the decoder portion of the VAE yields a likelihood distribution $p(x|z)$, and is parametrized on a neural network by weights collectively denoted ϕ. Hence we more properly denote the decoder portion of the VAE as $p_\phi(x|z)$. The output of the encoder are parameters of the latent distribution, which is sampled to yield the input into the decoder. Figure 4.2 shows a diagram of the structure of a VAE.

The Kullback-Leibler divergence between the approximate and the real posterior distributions is given by,

$$D_{KL}(q_\theta(z|x_i)||p(z|x_i)) = -\int q_\theta(z|x_i) log\left(\frac{p(z|x_i)}{q_\theta(z|x_i)}\right) dz \geq 0 \tag{4.4}$$

Then we can apply Bayes Theorem to the equation, getting,

$$D_{KL}(q_\theta(z|x_i)||p(z|x_i)) = -\int q_\theta(z|x_i) log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)p(x_i)}\right) dz \geq 0 \tag{4.5}$$

Now using simple logarithm properties, we get,

$$D_{KL}(q_\theta(z|x_i)||p(z|x_i)) = -\int q_\theta(z|x_i) \left[log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right) - log(p(x_i)\right] dz \geq 0 \tag{4.6}$$

Then we distribute the integrand,

$$-\int q_\theta(z|x_i) log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right) dz + \int q_\theta(z|x_i) log(p(x_i)) dz \geq 0 \tag{4.7}$$

We note that $log(p(x_i)$ is constant and can therefore be pulled out of the second integral above, then we get,

$$-\int q_\theta(z|x_i) log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right) dz + log(p(x_i)) \int q_\theta(z|x_i) dz \geq 0 \tag{4.8}$$

As $q_\theta(z|x_i)$ is a probability distribution, it integrates to 1. We get,

$$-\int q_\theta(z|x_i) log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right) dz + log(p(x_i)) \geq 0 \tag{4.9}$$

Then carrying the integral over the other side of the inequality, we get,

$$log(p(x_i)) \geq \int q_\theta(z|x_i) log\left(\frac{p_\phi(x_i|z)p(z)}{q_\theta(z|x_i)}\right) dz \tag{4.10}$$

Applying again logarithm properties, we get,

$$log(p(x_i)) \geq \int q_\theta(z|x_i) \left[log(p_\phi(x_i|z)) + log(p(z)) - log(q_\theta(z|x_i))\right] dz \tag{4.11}$$

Knowing that the right part of the inequality is an Expectation, we rewrite the inequiality as,

$$log(p(x_i)) \geq E_{q_\theta(z|x_i)} \left[log(p_\phi(x_i|z) + log(p(z)) - log(q_\theta(z|x_i)) \right] \tag{4.12}$$

$$log(p(x_i)) \geq E_{q_\theta(z|x_i)} \left[log(p(x_i, z) - log(q_\theta(z|x_i)) \right] \tag{4.13}$$

From Equation 4.10 it also follows that :

$$log(p(x_i)) \geq \int q_\theta(z|x_i) log \left(\frac{p(z)}{q_\theta(z|x_i)} \right) dz + \int q_\theta(z|x_i) log(p_\phi(x_i|z)) dz \tag{4.14}$$

$$log(p(x_i)) \geq -D_{KL}(q_\theta(z|x_i)||p(z)) + E_{q_\theta(z|x_i)}[log(p_\phi(x_i|z))] \tag{4.15}$$

The right part of the inequality is called the Evidence Lower Bound (ELBO) also known as the variational lower bound. The objective is to maximize the ELBO because this will maximize the log probability of our data by proxy. The Kullback-Leibler term is a regularizer that forces the latent space to have an specific form. The second term is called a reconstruction term because it is a measure of the likelihood of the reconstructed data output at the decoder.

We are free to choose what structure we want to get for our latent variables. So there is an interesting fact [36], if we choose a gaussian representation for the latent prior $p(z)$ and the approcimate posterior, $q_\theta(z|x_i)$, then we can obtain a closed form for the loss function.

Re-parametrization trick

In order to apply backpropagation we need a differentiable Expectation with respect $q_\theta(z|x)$. So we use the so-called reparametrization trick [35]. Let z be a continuous random variable, and $z \sim q_\theta(z|x_i)$ be some conditional distribution. It is then often possible to express the random variable z as a deterministic variable $z = g_\theta(\epsilon, x)$, where ϵ is an auxiliary variable with independent marginal $p(\epsilon)$, and $g_\theta(.)$ is some vector-valued function parameterized by θ.

Given the deterministic mapping $z = g_\theta(z|x)$ with $dx = \prod_i dx_i$ for infinitesimals, then,

$$q_\theta(z|x) \prod_i dz_i = p(\epsilon) \prod_i d\epsilon_i \tag{4.16}$$

Therefore,

$$\int q_\theta(z|x) f(z) dz = \int p(\epsilon) f(z) d\epsilon = \int p(\epsilon) f(g_\theta(\epsilon, x)) d\epsilon \tag{4.17}$$

So we can construct a differentiable estimator as,

$$\int q_\theta(z|x) f(z) dz \simeq \frac{1}{L} \sum_{l=1}^{L} f(g_\theta(\epsilon^l, x)) \tag{4.18}$$

where $\epsilon^l \sim p(\epsilon)$. In the Gaussian case, with $z \sim p(z|x) = \mathcal{N}(\mu, \sigma^2)$, a reparametrization could be $z = \mu + \sigma\epsilon$ in our VAE we use $z = \mu + e^{\sigma/2}\epsilon$, because we use the $log(\sigma)$ for numerical stability. Where ϵ is an auxiliary noise variable, $\epsilon \sim \mathcal{N}(0, 1)$. So,

$$E_{\mathcal{N}(z;\mu,\sigma^2)}\left[f(z)\right] = E_{\mathcal{N}(\epsilon;0,1)}\left[f(\mu + \sigma\epsilon)\right] \simeq \frac{1}{L}\sum_{l=1}^{L} f(\mu + \sigma\epsilon^l) \tag{4.19}$$

where $\epsilon^l \simeq \mathcal{N}(0, 1)$.

Gaussian VAE Loss

Choosing,

$$p(z) \to \frac{1}{\sqrt{2\pi\sigma_p^2}} exp\left(-\frac{(x-\mu_p)^2}{2\sigma_p^2}\right) \tag{4.20}$$

and,

$$q_\theta(z|x_i) \to \frac{1}{\sqrt{2\pi\sigma_q^2}} exp\left(-\frac{(x-\mu_p)^2}{2\sigma_q^2}\right) \tag{4.21}$$

then the Kullback-Leibler term in the ELBO changes to,

$$-D_{KL}(q_\theta(z|x_i)||p(z)) = \int \frac{1}{\sqrt{2\pi\sigma_q^2}} exp\left(-\frac{(x-\mu_p)^2}{2\sigma_q^2}\right) log\left(\frac{\frac{1}{\sqrt{2\pi\sigma_p^2}} exp\left(-\frac{(x-\mu_p)^2}{2\sigma_p^2}\right)}{\frac{1}{\sqrt{2\pi\sigma_q^2}} exp\left(-\frac{(x-\mu_p)^2}{2\sigma_q^2}\right)}\right) \tag{4.22}$$

Evaluating the $log(.)$ term simplifies the expresion into,

$$\begin{aligned} -D_{KL}(q_\theta(z|x_i)||p(z)) = \int \frac{1}{\sqrt{2\pi\sigma_q^2}} exp\left(-\frac{(x-\mu_p)^2}{2\sigma_q^2}\right) \times \\ \left[-\frac{1}{2}log(2\pi) - log(\sigma_p) - \frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{1}{2}log(2\pi) + log(\sigma_q) + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right] dz \end{aligned} \tag{4.23}$$

and this can be simplified into,

$$\begin{aligned} -D_{KL}(q_\theta(z|x_i)||p(z)) = \frac{1}{\sqrt{2\pi\sigma_q^2}} \int exp\left(-\frac{(x-\mu_p)^2}{2\sigma_q^2}\right) \times \\ \left[-log(\sigma_p) - \frac{(x-\mu_p)^2}{2\sigma_p^2}) + log(\sigma_q) + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right] dz \end{aligned} \tag{4.24}$$

and continuing with the simplification we obtain,

$$-D_{KL}(q_\theta(z|x_i)||p(z)) = \frac{1}{\sqrt{2\pi\sigma_q^2}} \int exp\left(-\frac{(x-\mu_p)^2}{2\sigma_q^2}\right) \times \left[log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right] dz \quad (4.25)$$

Expresing Equation 4.25 as an Expectation we get,

$$\begin{aligned} -D_{KL}(q_\theta(z|x_i)||p(z)) &= E_q\left[log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right] \\ &= log\left(\frac{\sigma_q}{\sigma_p}\right) + E_q\left[-\frac{(x-\mu_p)^2}{2\sigma_p^2} + \frac{(x-\mu_q)^2}{2\sigma_q^2}\right] \\ &= log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left[(x-\mu_p)^2\right] + \frac{1}{2\sigma_q^2}E_q\left[(x-\mu_q)^2\right] \end{aligned} \quad (4.26)$$

Since $\sigma_q^2 = E_q\left[(x-\mu_q)^2\right]$, it follow that,

$$\begin{aligned} -D_{KL}(q_\theta(z|x_i)||p(z)) &= log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left[(x-\mu_p)^2\right] + \frac{\sigma_q^2}{2\sigma_q^2} = \\ &= log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left[(x-\mu_p)^2\right] + \frac{1}{2} \\ &= log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left[(x-\mu_q+\mu_q-\mu_p)^2\right] + \frac{1}{2} \\ &= log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}E_q\left[(x-\mu_q)^2 + 2(x-\mu_q)(\mu_q-\mu_p) + (\mu_q-\mu_p)^2\right] + \frac{1}{2} \\ = log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}&\left(E_q\left[(x-\mu_q)^2\right] + 2E_q\left[(x-\mu_q)(\mu_q-\mu_p)\right] + E_q\left[(\mu_q-\mu_p)^2\right]\right) + \frac{1}{2} \\ &= log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{1}{2\sigma_p^2}\left(\sigma_q^2 + 2*0*(\mu_q-\mu_p) + (\mu_q-\mu_p)^2\right) + \frac{1}{2} \\ &= log\left(\frac{\sigma_q}{\sigma_p}\right) - \frac{\sigma_q^2 + (\mu_q-\mu_p)^2}{2\sigma_p^2} + \frac{1}{2} \end{aligned} \quad (4.27)$$

And when we take $\sigma_p = 1$ and $\mu_p = 0$, we get,

$$\begin{aligned} -D_{KL}(q_\theta(z|x_i)||p(z)) &= log(\sigma_q) - \frac{\sigma_q^2 + \mu_q^2}{2} + \frac{1}{2} \\ &= \frac{1}{2} log(\sigma_q^2) - \frac{\sigma_q^2 + \mu_q^2}{2} + \frac{1}{2} \\ &= \frac{1}{2}\left[1 + log(\sigma_q^2) - \sigma_q^2 - \mu_q^2\right] \end{aligned} \tag{4.28}$$

Remembering the ELBO, Equation 4.15,

$$log(p(x_i)) \geq -D_{KL}(q_\theta(z|x_i)||p(z)) + E_{q_\theta(z|x_i)}[log(p_\phi(x_i|z))] \tag{4.29}$$

From which it follows that the contribution from a given datum xi and a single stochastic draw towards the objective to be maximized is,

$$\frac{1}{2}\left[1 + log(\sigma_j^2) - \sigma_j^2 - \mu_j^2\right] + E_{q_\theta(z|x_i)}[log(p_\phi(x_i|z))] \tag{4.30}$$

where σ_j^2 and μ_j are parameters into the approximate distribution, q, and j is an index into the latent vector z, J is the dimension of the latent vector z, and L is the number of samples stochastically drawn according to the re-parametrization trick.

$$\mathcal{G} = \sum_{j=1}^{J} \frac{1}{2}\left[1 + log(\sigma_j^2) - \sigma_j^2 - \mu_j^2\right] + \frac{1}{L}\sum_{l}^{L} E_{q_a(z|x_i)}\left[log(p(x_i|z_{i,l}))\right] \tag{4.31}$$

Equation 4.31 is mean to be maximazized during training, so the opposite will be the loss function that we are searching, as,

$$\mathcal{L} = -\sum_{j=1}^{J} \frac{1}{2}\left[1 + log(\sigma_j^2) - \sigma_j^2 - \mu_j^2\right] - \frac{1}{L}\sum_{l}^{L} E_{q_a(z|x_i)}\left[log(p(x_i|z_{i,l}))\right] \tag{4.32}$$

Chapter 5

Methodology

5.1 Facial Action Units System (FACS)

5.1.1 What is FACS ?

FACS is a system to taxonomize human facial movements by their appearance on the face, this codifying system is based on a system developed by a Swedish anatomist called Carl-Herman Hjorsjo, later on it was published by the psychologist Paul Ekman and his collegue Wallace V. Friesen in 1978 [37]. An update of this work was published in 2002 improving the previous results. FACS basically makes a classification of individual muscle movements or group muscle movements, this classifiaction is divided in different Action Units(AU), and different combinations of AUs codify different emotions. For example if a face has active AUs like 7,6,12 then we can say that this face is a smilling face. This system is really useful for psychologist but also for computer vision systems that want to analyze human behaviour, as thanks to this system we have a methodology to describe emotions and their intensity.

5.1.2 Action Units (AU)

Action Units are the fundamental actions of individual muscles or groups of muscles.AUs are the fundamental actions of individual muscles or groups of muscles. As AUs markers are independent of any interpretation, they can be used for any higher order decision making process including recognition of basic emotions, or pre-programmed commands for an ambient intelligent environment. The FACS Manual is over 500 pages in length and provides the AUs, as well as Ekman's interpretation of their meaning. There are more than 100 different AUs. An example of some AUs and their combination can be checked in Figure 5.1. AUs are useful for all kind of tasks that involve facial expressions such as computer facial animation[38], computer processing of body language[39], facial electromyography[40], analysis of depression[41] or measurement of pain in patients unable to express themselves verbally[42].

Figure 5.1: AU examples. Image extracted from OpenFace wiki

AUs are also classified by their intensity, as some of them can be very similar this is important to distinguish between them. The intensity is denoted by letters, the meaning of each of them is the following :

- A, trace
- B, slight
- C, marked or pronounced
- D, severe or extreme
- E maximun

AUs can also appear laterally o bilaterally so they usually get complemented with a letter denoting where it appears, R (right), L (left).

AUs are very useful for face generation tasks. In general they allow us to differenciate certain emotions. It is not a perfect system as sometimes depending on the ethnicity or the face geometry some emotions are difficult to distinguish. For our task we needed faces were AUs intensity is always under a very little threshold, as we wanted to generate neutral faces.

5.2 Neutral Face Generation from frontal faces

Our main objective is to generate a neutral face from frontal faces of the same subject. For this we needed a subject independent model. This means that the model should learn how to identify relevant features from the frontal faces and codify them. VAE fits this requirements.

Variational Autoencoders (VAE's) can learn[35] the distribution of datasets and adapt them to certain distribution functions, as gaussian in our case. This allows us to get a meaningful codification for our dataset. VAE learns the features of our dataset and creates a latent space related to it. As bigger and more diverse our dataset is, the larger and more diverse our latent space will be.

Gathering frontal faces plus a neutral faces of the same subjects is a hard job. The easiest way is sampling from videos. Using the AU classification system we can distinguish neutral faces from other emotions. But when people talk is difficult to obtain neutral faces and easy to get blurry images.

Training with small datsets usually gives poor results, as the network will get quickly overfitted. Our approach to solve this is to apply transfer learning[43].

For applying transfer learning we first train a network specialized in generating neutral faces. We just need a dataset containing neutral faces. This dataset is easier to gather.

Then we can apply transfer learning to the final generator in two ways that are totally combinable :

- Attaching the pre-trained decoder and freezing its variables.
- Adding a regularization term.

5.3 Implementation

5.3.1 Some general insights of our network

The basis structure of our networks is easy to understand. In the beginning we use some Convolutional layers as they are able[44][45] to extract features from images.

Then we need to gather this features, filter them or give them another meaning. This task will be performed by a multilayered network.

For the latent space we use 2 layers, one for the mean, and another for the standard deviation. We combine this 2 outputs as $z = \mu + e^{\sigma/2}\epsilon$, because we use the $log(\sigma)$ for numerical stability, ϵ is a variable sampled from $\mathcal{N}(0,1)$ for applying the re-parametrization trick.

5.3.2 Architectures

The basic structure of our first networks can be seen in Figure 5.2 . The input will be a neutral face and the output it's reconstruction. In the middle the latent space will learn the distribution of our dataset.

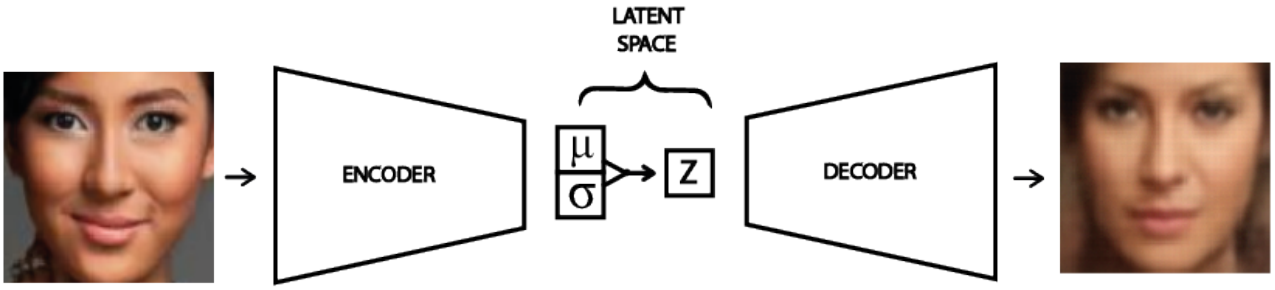


Figure 5.2: Neutra Face Generator schema

VGG Structure				
Layer type	Input size 2 Code	Output size	Filters	Kernel size
Convo	(None,128,128,3)	(None,128,128,64)	64	3x3
Convo	(None,128,128,3)	(None,128,128,64)	64	3x3
Max-Pool	(None,128,128,3)	(None,64,64,128)	128	2x2
Convo	(None,64,64,128)	(None,64,64,128)	128	3x3
Convo	(None,64,64,128)	(None,64,64,128)	128	3x3
Max-Pool	(None,64,64,128)	(None,32,32,256)	256	2x2
Convo	(None,32,32,256)	(None,32,32,256)	256	3x3
Convo	(None,32,32,256)	(None,32,32,256)	256	3x3
Convo	(None,32,32,256)	(None,32,32,256)	256	3x3
Max-Pool	(None,32,32,256)	(None,16,16,512)	512	2x2
Convo	(None,16,16,512)	(None,16,16,512)	512	3x3
Convo	(None,16,16,512)	(None,16,16,512)	512	3x3
Convo	(None,16,16,512)	(None,16,16,512)	512	3x3
Max-Pool	(None,16,16,512)	(None,8,8,512)	512	2x2
Flatten	(None,8,8,512)	(None,32768)	-	-
FC	(None,32768)	(None,4096)	-	-
FC	(None,4096)	(None,4096)	-	-
FC no Drop	(None,4096)	(None,1024)	-	-

Table 5.1: VGG schema

For the Neutral Face Generation we tried 3 different structures. As we wanted to test their perfomance in this task. This structures are :

- Custom Visual Geometry Group (VGG) : VGG structure is characterized for using small kernel sizes for the convolutional units. This allows the network to have a deeper structure. Table 5.1 show an schema of tour custom VGG encoder. The decoder is just the inverse of the encoder.

- Custom Residual Network (ResNet) : Residual Networks use skip connections or shortcuts to jump over some layers. This means that output of lets say layer 2 can be added to the input of layer 4 for example. The motivation behind this procedure is avoiding the vanishing gradient problem. ResNet's have shown to be very useful in image recognition[46]. Table 5.2 shows our implementation. We just show the first residual block, this structure is repeated until get an image of dimension (None,8,8,512). The decoder is just the inverse of the encoder.

- Big Convolutional Network (BigConvNet) : This implementation doesn't follow any known model. We just tried a different structure from VGG or ResNet. Table 5.3 shows its structure. The decoder is just the inverse of the encoder.

VGG Structure				
Layer type	Input size 2 Code	Output size	Filters	Kernel size
Convo0	(None,128,128,3)	(None,64,64,64)	64	7x7
Convo1	(None,64,64,64)	(None,64,64,64)	64	3x3
Convo2	(None,64,64,64)	(None,64,64,128)	64	3x3
Convo3	Convo2+Convo0	(None,64,64,128)	64	3x3
Convo4	(None,64,64,64)	(None,64,64,64)	64	3x3
Convo5	Covo4+Convo2	(None,32,32,128)	128	3x3
-	-	-	-	-
Flatten	(None,8,8,512)	(None,32768)	-	-
FC	(None,32768)	(None,4096)	-	-
FC	(None,4096)	(None,4096)	-	-
FC no Drop	(None,4096)	(None,1024)	-	-

Table 5.2: ResNet schema

VGG Structure				
Layer type	Input size 2 Code	Output size	Filters	Kernel size
Convo	(None,128,128,3)	(None,128,128,16)	16	3x3
Max-Pool	(None,128,128,16)	(None,64,64,16)	16	2x2
Convo	(None,64,64,16)	(None,64,64,32)	32	3x3
Max-Pool	(None,64,64,32)	(None,32,32,32)	32	2x2
Convo	(None,32,32,32)	(None,32,32,64)	64	3x3
Max-Pool	(None,32,32,64)	(None,16,16,64)	64	2x2
Convo	(None,16,16,64)	(None,16,16,128)	128	3x3
Max-Pool	(None,16,16,128)	(None,8,8,128)	128	2x2
Convo	(None,8,8,128)	(None,8,8,256)	256	3x3
Max-Pool	(None,8,8,256)	(None,4,4,256)	256	2x2
Convo	(None,4,4,256)	(None,4,4,512)	512	3x3
Convo	(None,4,4,512)	(None,4,4,512)	512	3x3
Convo	(None,4,4,1024)	(None,4,4,1024)	1024	3x3
Convo	(None,4,4,1024)	(None,4,4,1024)	1024	3x3
Flatten	(None,4,4,1024)	(None,16384)	-	-
FC	(None,16384)	(None,4096)	-	-
FC	(None,4096)	(None,4096)	-	-
FC no Drop	(None,4096)	(None,1024)	-	-

Table 5.3: BigConvNet schema

To generate neutral faces from different frontal faces of a same subject we used a different structure. As the frontal images are not labeled the input of this network is always random. It didn't had sense to use specialized encoders for each of the images. Instead we share the same encoder between the input images (frontal faces). This is called weight sharing, so if there are 3 input images they will all be processed by the same layers. Then we will be training only one encoder, not 3. Figure 5.3 illustrates how this is implemented. For this part we used the networks structures that generated better neutral faces.

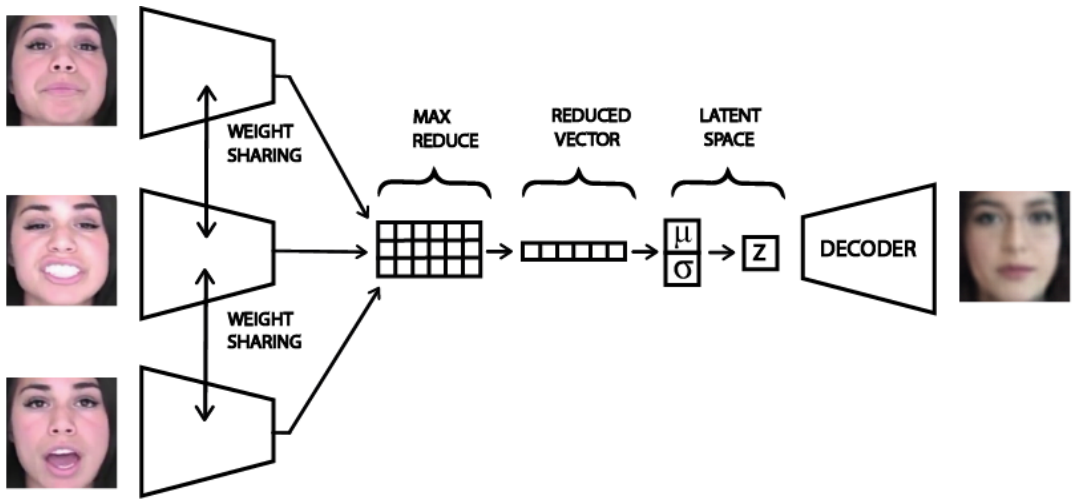


Figure 5.3: AU examples

5.3.3 Train methodology

The dataset was divided in Training/Validation with a proportion 90/10 approximately. The training stops when Validation Error is too different from Training Error, in our case a 30%.

Loss function of the Neutral Face Generator

For this part the loss function involves 2 terms. The first one,

$$Reconstruction\ loss = (X_T - X_R)^2 \tag{5.1}$$

were X_T is the input image, X_R is the reconstructed image. This term mesuares the reconstruction error. As images are componed of continous points we used the Mean Squared Error (MSE) as measure.

The second term,

$$KL\ loss = -\frac{1}{2}\left[1 + \sigma - e^{\sigma} - \mu^2\right] \tag{5.2}$$

were σ is the logarithm of the computed standard deviation and μ is the computed mean for each image.

So the loss function is,

$$\mathcal{L} = (X_T - X_R)^2 - \frac{\beta}{2}\left[1 + \sigma - e^{\sigma} - \mu^2\right] \tag{5.3}$$

were β is a variable that we have tuned in 2 different ways :

- Incremental β-VAE : β variable gets incremented from 0.001 to 1 every 5 epochs if the reconstruction loss is under a certain threshold (10.0) .
- β-VAE : as some papers[47][48] show using a higher value for the regularization term forces the latent space to learn more features from the dataset. This value is fixed on all epochs. Best results were achived with $\beta = 2$

$$\mathcal{L} = (X_T - X_R)^2 - \frac{\beta}{2}\left[1 + \sigma - e^{\sigma} - \mu^2\right] \tag{5.4}$$

Loss function of the Neutral Face Generator from frontal faces

For this part we don't the KL regularization term. As we don't need to generate samples from a latent space. We added a regularization term involving the generated vectors obtained by passing the neutral faces of the second dataset into the Neutral Face Generator created in the first part, we denote this vectors as V_T. We use a simple loss function to compare this vectors to the ones that we obtain, we denote this vectors as V_R. The idea is that this regularization term forces the new latent space to have a similar structure that the previous generator.

The new regularization term is,

$$\textit{Regularization loss} = |V_T - V_R| \tag{5.5}$$

So the loss function is,

$$\mathcal{L} = (X_T - X_R)^2 + \lambda|V_T - V_R| \tag{5.6}$$

were we have tested λ with 2 different values, 0 and 0.1. We used a low value as the vectors used to compute this loss are generated using the re-parametrization trick and this makes them noisy.

Chapter 6

Experiments and results

6.1 Setups

6.1.1 TensorFlow, Scikit-learn and Openface

TensorFlow

TensorFlow is a Python-friendly open source library for numerical computation. It's a symbolic math library and is commonly used for machine learning applications such as neural networks. It is developed by the Google Brain team for internal use, but finally they released it as an open-source library, so anyone around the world can learn or develop machine learning in a easy way.

TensorFlow includes lots of optimized algorithms and structures that facilitate the job. TensorFlow also allows to run programs in CPU and GPU (with also CUDA extensions for high performance tasks).

Early versions of TensorFlow were very useful for experts or people with some knowledge in machine learning but maybe a little bit messy for beginners, that's why the new versions TensoFlow2.0 is integrated with Keras that simplifies a lot several tasks.

In our project we have used TensorFlow-GPU 1.14 as we were going to train a big network we needed to run the computations in GPU for reducing training time. This version is adapted to run with CUDA drivers so it is optimized for Nvidia GPU's, as the ones that are present in the server that we used.

TensorFlow has an API's for Python, C++, Haskell, Java, Go and Rust, and there are also not oficial librarys for C, Julia, R Scala and OCaml.

Scikit-learn

Scikit-learn is a free software machine learning library for Python. It has various classification, regressin and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and T-SNE.

For our project we used the T-SNE implementation. T-SNE is a machine learning algorithm for visualization. It is a nonlinear dimensionality reduction technique for embedding high dimensional data for visualization in a low-dimensional space of two or three dimensions. This algorithm is presented in a library called sklearn. This allows us to show the distribution of our data.

OpenFace

OpenFace is project [49] developed by Tadas Baltrusaitis in collaboration with CMU MultiComp Lab led by Prof. Louis-Philippe Morency. This project is a tool intended for computer vision and machine learning researchers, affective computing community and people interested in building interactive applications based on facial behavior analysis. OpenFace is the first toolkit capable of facial landmark detection, head pose estimation, facial action unit recognition,and eye-gaze estimation with available source code for both running and training the models. The computer vision algorithms which represent the core of OpenFace demonstrate state-of-the-art results in all of the above mentioned tasks.

Figure 6.1: OpenFace ability to recognise facial landmarks. Image extracted from OpenFace wiki.

Using this software allows you to extract a lot of features from the images, it also have some variants, you can choose to present video or images, you can even choose between a sequence of related images (frames of a video) or to present isolated images. Some functionalities are :

- Facial Landmark Detection
- Facial Landmark and head pose tracking [50] [51]
- Facial Action Unit Recognition [52]
- Gaze tracking [53]
- Facial Feature Extraction (aligned faces and HOG features)

This software is able to recognise 68 facial landmarks, this gives a very precise idea of the face. It also allows to crop the face using this landmarks and align them. It also provides a .csv with a detalied analysis of each frame or image that can be used to filter images, in this .csv there is information as : confidence, eye gaze, pose in mm and rotation, landmarks in 2-D and 3-D, facial action units with presence and intensity.

6.1.2 Hardware

The hardware used to carry out this work is :

- For the warm-up with TensorFlow and the preliminary tests with MNIST a MSI CX62 6QD was used, with the following technical specifications :
  - Processor : Intel Core i7-6700HQ (2.7 GHz, 6 MB)
  - RAM : 8GB DDR4 SODIMM
  - GPU : NVIDIA 940MX
- For the last training the networks it was needed a higher computing capacity as training times in the laptop where higher than 7 hours, so a 10-GPU server of the Computer Vision Center located in the Autonomous University of Barcelona (UAB) was used with 9 NVIDIA GeForce GTX 1080 Ti (12GB) and a Intel Quadro P6000 (24GB).

6.2 Warm-up

To practice with VAE structure and hyperparameters the first step of the project was to develop from scratch a VAE of handwritten digits instead of directy start with the neutral face generation. We used the Modified National Institute of Standars and Technology (MNIST) dataset, as it is one of the reference datasets in image recognition or generation for testing performance.

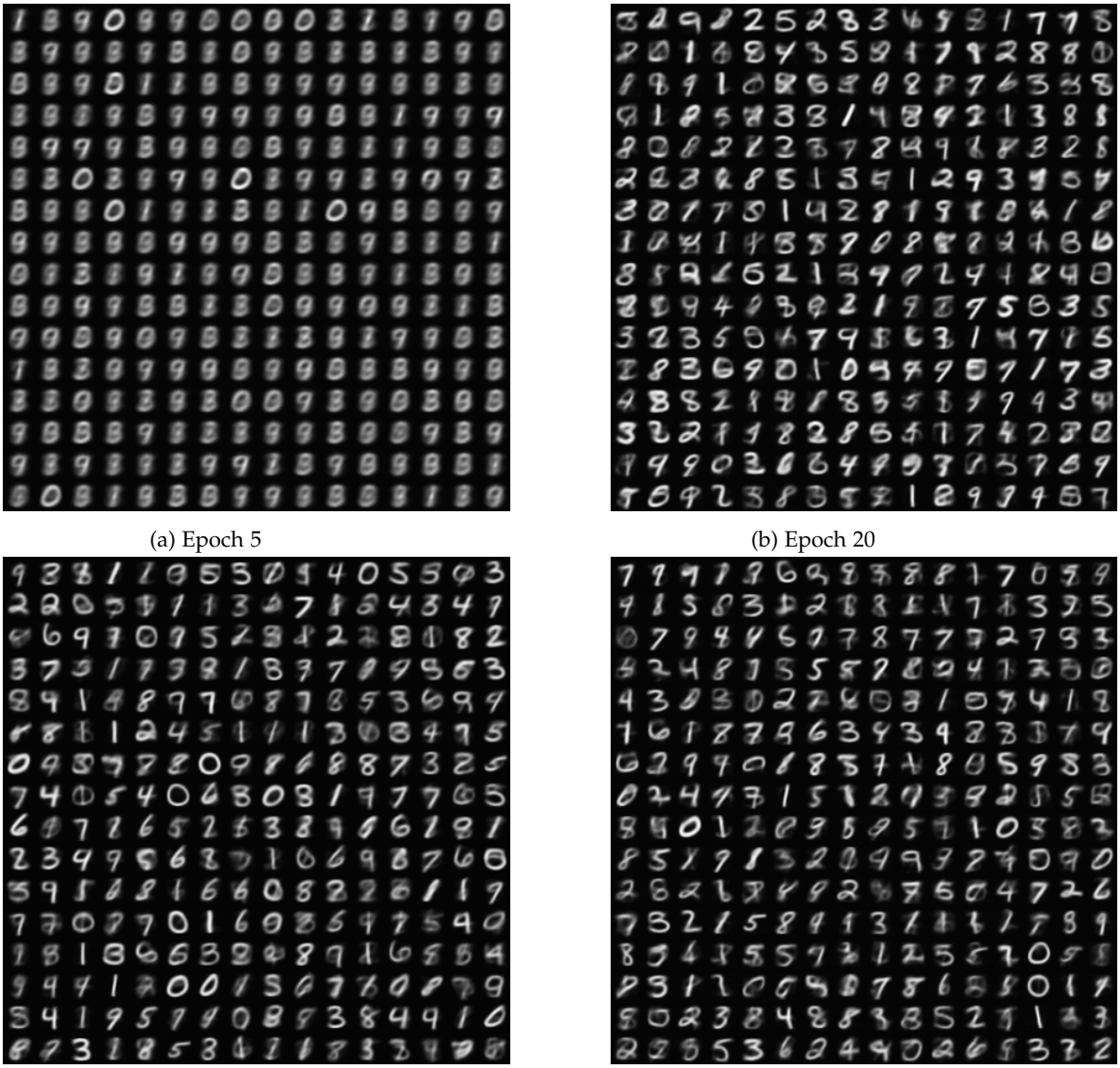
The preprocessing of the data was quite simple, just normalizing the value of each pixel to have values between 0 and 1, as NN are very sensitive to high scales.

(a) Original Image (b) Reconstruction latent dim 25 (c) Original Image (d) Reconstruction latent dim 10

Figure 6.2: Different reconstructions depending on the latent dimension

Tuned hyperparameters were: dropout rate, learning rate and latent dimension. In Figures 6.2 (a) and 6.2 (b) we can observe the original and the reconstruction of a VAE with a latent space of 25 variables. In Figures 6.2 (c) and 6.2 (d) we observe the reconstruction of a VAE with a latent space of 10 variables.

Reconstruction works better with a latent space of 10. We also checked that better reconstruction gives us a better generation. So using a VAE with a 10 dimensional latent space we obtain the converngece showed un Figure 6.3.



(a) Epoch 5

(b) Epoch 20

(c) Epoch 30

(d) Epoch 50

Figure 6.3: Example of convergence of a MNIST latent space

From this experiments we concluded that VAE works better with low dimensional latent space. The optimal learning rate was 1e-04 using the optimizer Adam. Dropout was useful to avoid overfitting, using it in the Multilayered hidden layers gived us better results when evaluating in the validation set.

6.3 The datasets

6.3.1 Data basic information

For the project we used 2 different datasets. As the first part of the project consisted in generating neutral faces, the first dataset consists on approximately 5500 frontal neutral faces extracted from 2 different datasets. CelebA [54],componed of 200K images of celebrities and a little dataset provided by the University of Essex called faces94, componed by 395 subjects, where the most part are students between 18-20 years old. This dataset has a very nice ethnical and gender diversity.

The second dataset consists on several random frontal faces and a neutral face of the same subject. This images were extracted from the First Impressions V2 (CVPR'17)[55] dataset. This dataset is composed of 10000 clips with an average duration of 15s extracted from 3000 different YouTube HD videos of people facing and speaking in English to a camera. People in the videos show different gender, age, nationality, and ethnicity. This second dataset is componed of 608 subjects.

6.3.2 How was the data gathered ?

The problem with CelebA and faces94 is that the most part aren't frontal or neutral faces. Here is were Openface became critical. Thanks to the Feature extraction project we were able to generate a .csv for each image with information as the pose, the AU markers or the facial landmarks, for a more developed project is also possible to extract the position of the eyes,

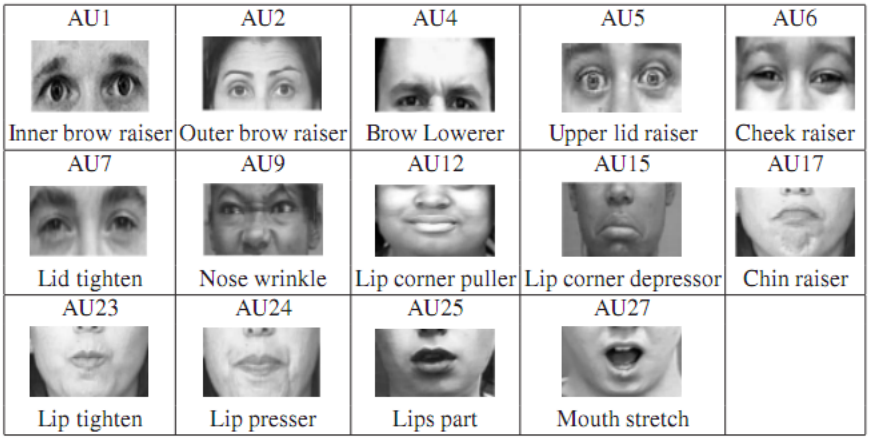


Figure 6.4: Checked AUs

but we feel that it wasn't necesary for the objectives of the project. To filter the faces into 5.5K frontal faces we developed a python script that reads the .csv of each image and check if the pose is frontal, this can be done in 2 ways, using the pose estimation given by Openface or using the landmarks to approximate the position of the face in relation with the camera,

both ways were explored giving better results just and check if the pose given by Openface. The next step was to check that the face is neutral. For this, first is important to check a confidence mesure that Openface gives, as sometimes the classificator makes mistakes. With a confidence level higher than 0.9 estimations can be considered as true. Once this is checked we take a look to some key AU markers that shouldn't be present in a neutral face. For this Openface provides 2 mesures, if an attribute is present and it's intensity. We decided to set a threshold to the intensity of the AU markers. The AU markers checked can be seen in Figure 6.4. In spite of we checked the AUs not all the faces are totally neutral, so there is some noise in this dataset. Figues 6.5 and 6.6 show filtered images from CelebA and faces94. The size of this dataset is 120 MB

Figure 6.5: CelebA examples

Figure 6.6: Essex examples

The second dataset was gathered from a video collection using the tools provided by Openface. The videos were passed one by one (using a bash script) to the Openface Feature Extraction project, which is able to generate a .csv for each video. This .csv's contain information about the face in each frame. Openface also crops each face frame in a directory. Once we gathered all the .csv. We used another python script pretty similar to the one used in the first part. In this case the script checks the .csv searching for neutral and frontal faces. To do this it checks for neutral faces using the same AU's as in the first part. For frontal faces it checks the rotation of the head. Then it extracts randomly one neutral face and 7 frontal faces. It would have been better to extract the most neutral face and very different expresions of the face, but is difficult to quantify this only with the information provided by Openface. That is why we decided to do it randomly. This dataset was manually checked. The size of this dataset is 126MB.

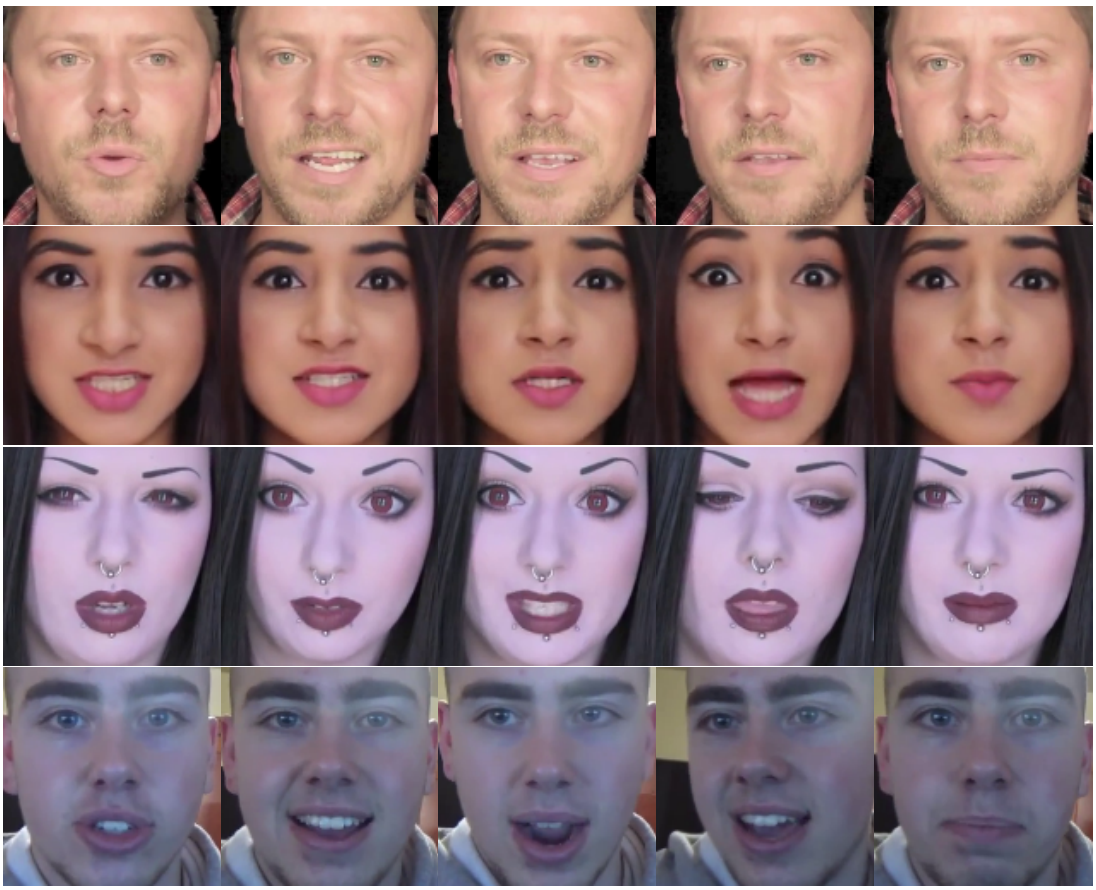


Figure 6.7: Second dataset examples

6.4 Neutral Face Generator

6.4.1 Hyperparameters

After several test we concluded that the best hyperparameters for this networks were :

- Learning Rate: 1e-05
- Dropout Rate: 0.75
- Activation functions: ELU
- Latent dimension : 64
- Batch size : 64
- Optimizer : Adam

6.4.2 Neutral Face Generator networks results

In this section we will show the obtained results and discuss some important aspects that can be deduced from them.

We noticed that in early epochs all the networks tend to generate femenine faces as can be seen in Figure 6.8. This makes us think that our first dataset is biased. It seems to present more women than men. We cannot see any young children or old people in our generations, despite the fact that there are some of them in the dataset. Our generations are always in an age range of 18 - 45 approximately. Our deduction is, first women frontal faces are more common in CelebA dataset as they are extracted from models or actresses and the percentage of photos of this style from women is higher than from men. Second, celebrities are usually in the same range of age (18 - 45) as our face generations and our other dataset faces94 is componed by subjects between 18 - 20 years old.

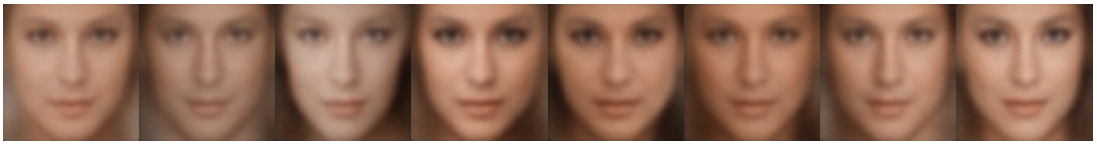


Figure 6.8: Pre-convergence faces

Now we present a table showing the minimum Mean Squared Error (MSE) achivied in the validation set , the epoch when it was achieved and the total epochs performed by each network. Both network were trained in 2 different ways, with incremental β and with $\beta = 2$.

As we can see in Table 6.1 training with $\beta = 2$ takes longer and offers worse minimum MSE in each of the networks. This makes sense as a higher β values pressure more the latent space to learn features of our dataset but decrease it's reconstruction power. Other important fact is that VGG trained with Incremental β training offers the best results for MSE reduction.

Network	Trainig style	Minimum MSE	Epoch of min MSE	Total Epochs
BigConvNet	$\beta = 2$	9.17	455	475
BigConvNet	Incremental β	8.75	300	355
ResNet	$\beta = 2$	7.67	245	305
ResNet	Incremental β	6.15	190	260
VGG	$\beta = 2$	7.05	355	475
VGG	Incremental β	5.41	315	425

Table 6.1: Minimum MSE, Epoch were its achived and Total Epochs of each network with different training styles

Now we will analyze subject by subject the network's visual reconstruction performance. Images can be checked in Figure 6.9.

In all the networks the first subject preserved the gender, except BigConvNet with $\beta = 2$ training, . Only ResNet with $\beta = 2$ training preserved correctly the beard. VGG preserved the beard but is difficult to see it. In general VGG with Incremental β training preserved better the face geometry and the skin color.

In all the networks the second subject preserved the gender but the ethnicity is better preserved by VGG with Incremental β training.

In all the networks the third subject preserved the gender. Only ResNet with both training styles and VGG with Incremental β training preserved the beard, but clearly the reconstrucion of VGG with Incremental β training is visually better.

For the fourth subject only ResNet and VGG with $\beta = 2$ training preserved the gender. None of the networks preserved the glasses. None of the networks seem to recognise correctly this face structure.

The fifth subject is interesting as it is an old man. All the networks preserved the gender. None of the networks was able to preserve the age. VGG with $\beta = 2$ training seemed to recognise the wrinkles as beard.

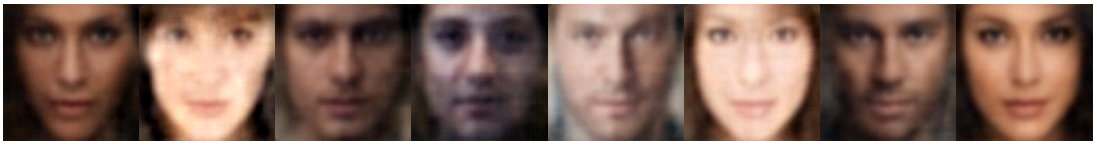
The sixth subject is an old woman. All the networks preserved the gender but any of them preserved the age. The most accuarate reconstruction was again performed by VGG with Incremental β training.

In all the networks the seventh subject preserved the gender. All the networks reconstructed the little beard that the subject presents. The best face geometry reconstruction was performed by VGG with Incremental β training.

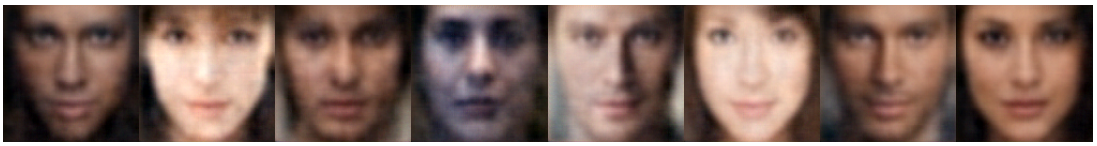
In all the networks the last subject preserved the gender. None of the networks preserved the cheeks make-up. ResNet with $\beta = 2$ training preserved the make-up of the eyes. The best reconstruction was performed by VGG with Incremental β training.

We conclude as Table 6.1 was announcing that the best reconstruction is performed by VGG with Incremental β training. It seems to recognise better the face geometry.

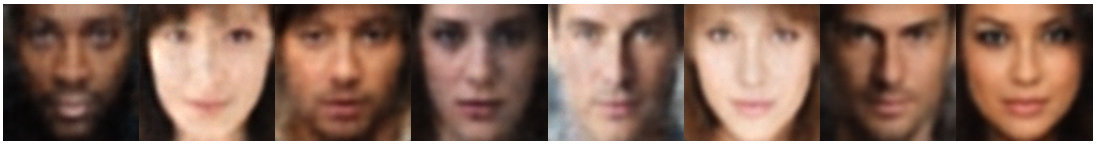
(a) Original Images



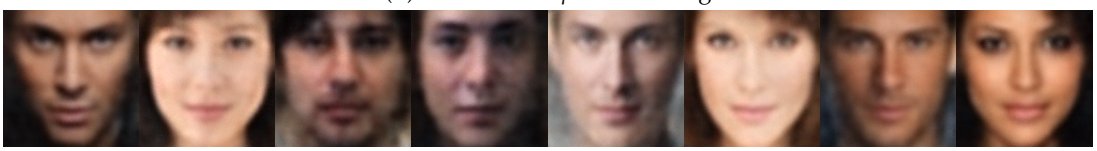
(b) BigConvNet with $\beta = 2$ training



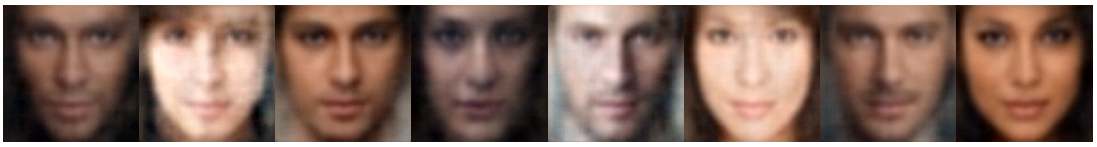
(c) BigConvNet with Incremental β training



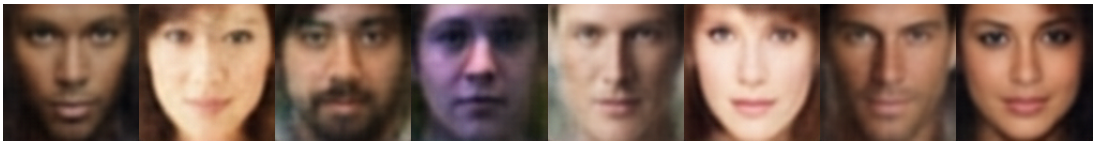
(d) ResNet with $\beta = 2$ training



(e) ResNet with Incremental β training



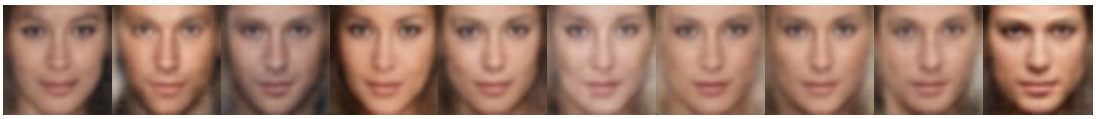
(f) VGG with $\beta = 2$ training



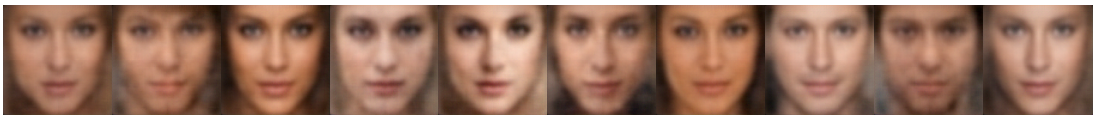
(g) VGG with Incremental β training

Figure 6.9: Reconstruction Images compared with the originals of each of the networks. First row shows the original images, next rows show the recontruction performed by the networks.

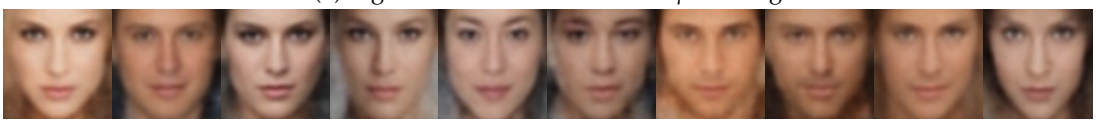
Now we present different face generations performed by our networks. This ace generations are obtained sampling directy from the latent space. All our networks converge to a latent space with distribution $\mathcal{N}(0,1)$. We just sampled from this distribution random vectors of dimension 64 and introduced them into the decoder.



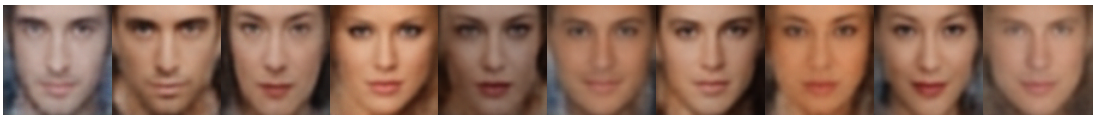
(a) BigConvNet with $\beta = 2$ training



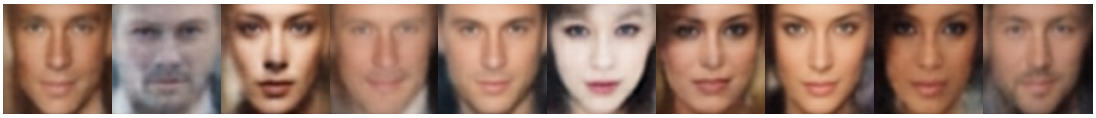
(b) BigConvNet with Incremental β training



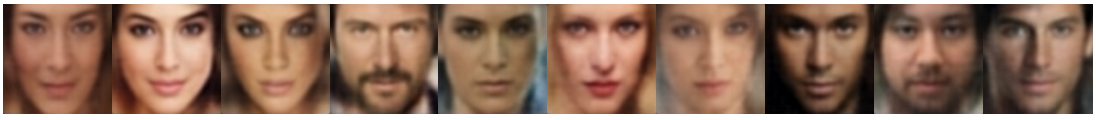
(c) ResNet with $\beta = 2$ training



(d) ResNet with Incremental β training



(e) VGG with $\beta = 2$ training



(f) VGG with Incremental β training

Figure 6.10: Generated images from the latent space of each of the network. The rows show the generation performed by the networks with different training styles.

As Figure 6.10 shows we cannot see a lot of diversity from the 3 first networks (Figure 6.10 (a), (b) and (c)). Skin color changes a little bit and gender also, but male faces still look a bit femenine. The 3 next networks show more diversity.

ResNet with Incremental β training (Figure 6.10 (d)) shows more masculine faces, like the first 2 subjects. We can oberve that the fifth subject shows a illumination variation.

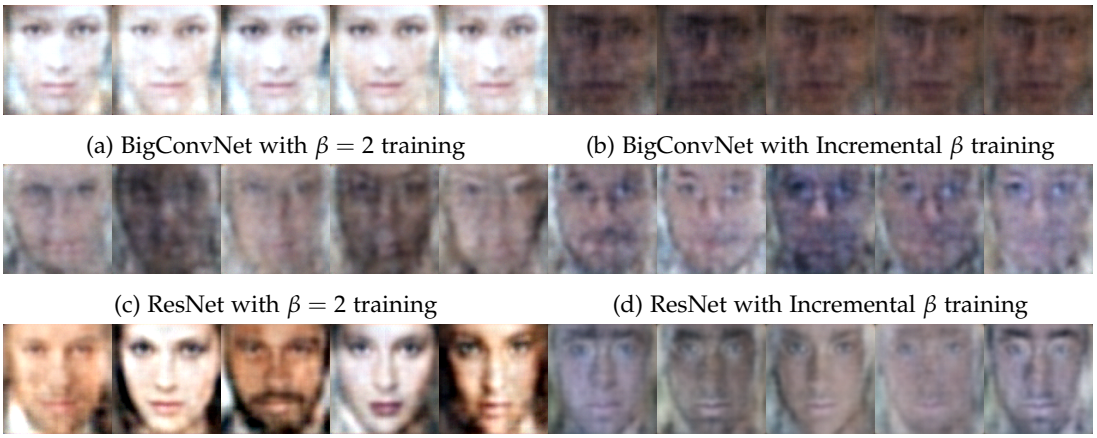
VGG with $\beta = 2$ training (Figure 6.10 (e)) shows more diversity in male face and ethnicity,

subject's sixth and eight show this variations. We observe the presence of beard in the second and last subjects. The third subject shows a very different face geometry that we cannot see on the previous networks. This network doesn't seem to show variety of illumination.

The best generation results are obtained in VGG with Incremental β training as Figure 6.10 (f) shows . Ethnicity diversity can be oberved on subject's first, fifth, sixth and eight. We can also see a very good beard generation on the fourth subject. Male and female faces are very different between them. We can observe illumination changes in the third, fifth, eight and last subjects.

In conclusion VGG with Incremental β training (Figure 6.10 (f)) offered the best generation results. It shows different face geometries combined with variety of skin colors and illumination.

The next face generations are obtained by introducing into the encoder random noisy images (this means we sample pixels from a $\mathcal{U}(0,1)$ distribution). The objective of this type of sampling is checking how good the distribution of the data is learned by the networks



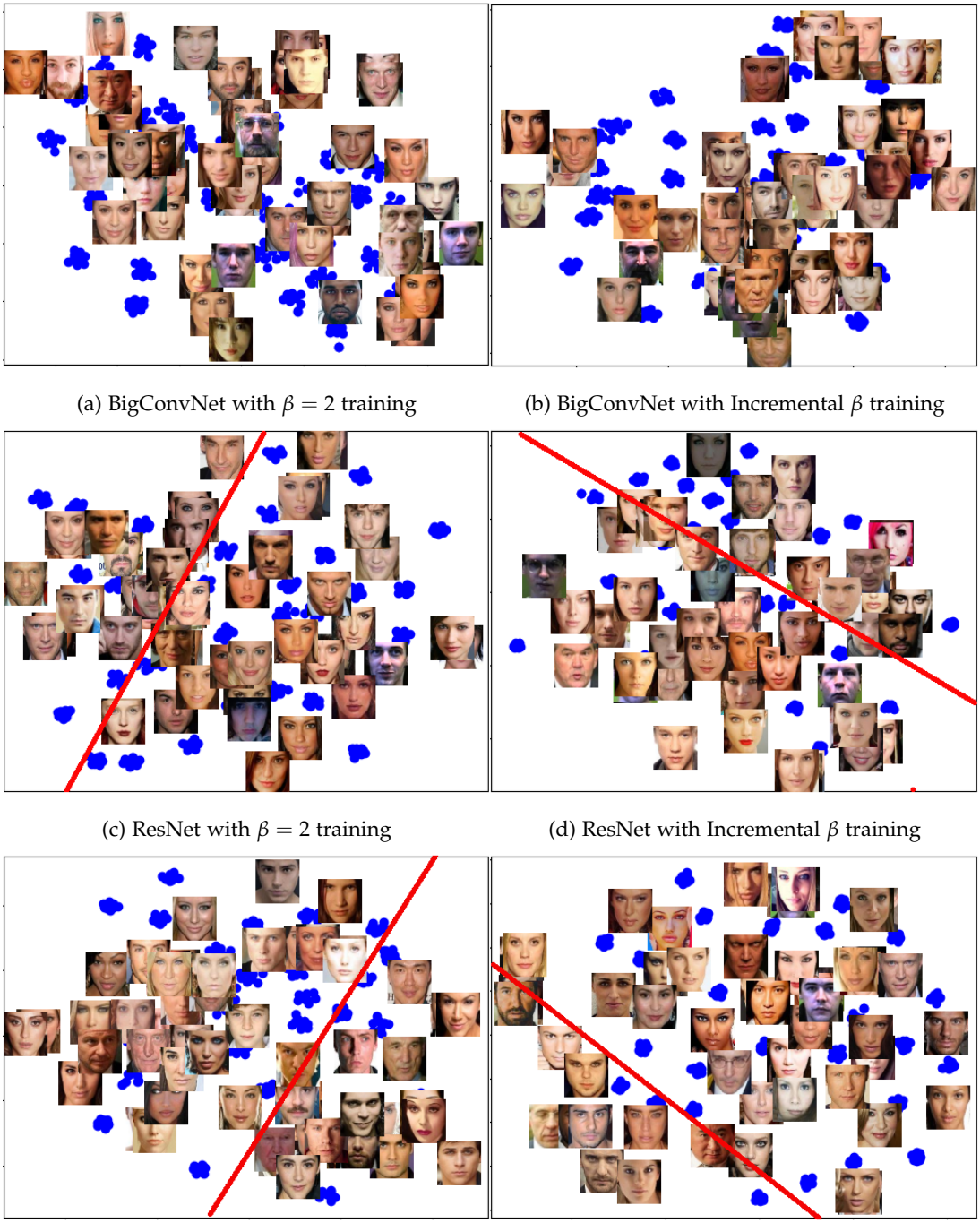
(a) BigConvNet with $\beta = 2$ training (b) BigConvNet with Incremental β training

(c) ResNet with $\beta = 2$ training (d) ResNet with Incremental β training

(e) VGG with $\beta = 2$ training (f) VGG with Incremental β training

Figure 6.11: Images generated from random noise sampled from $\mathcal{U}(0,1)$ distribution.

The first 3 networks doesn't seem to generate different faces when we sample fromrandom noise images (this corresponds to Figure 6.11 (a), (b) and (c)). In Figure 6.11 (d) we can see that the noisy faces show some variation. Figure 6.11 (e) corresponding to VGG with $\beta = 2$ training shows how this network learned so hard the data distribution that is able to generate noisy faces from random uniform noise. This is related with the pressure that $\beta = 2$ parameter makes on the creation of the latent space. Figure 6.11 (f) corresponding to VGG with Incremental β training shows that this network learned the data distribution, but wasn't able to add color correctly to the noisy faces. We conclude that $\beta = 2$ pressures the latent space to learn better the features of our dataset, but as we can observe on Figure 6.9 (f) and in Table 6.1 this affects the reconstruction performance of the networks.



(a) BigConvNet with $\beta = 2$ training

(b) BigConvNet with Incremental β training

(c) ResNet with $\beta = 2$ training

(d) ResNet with Incremental β training

(e) VGG with $\beta = 2$ training

(f) VGG with Incremental β training

Figure 6.12: Example of convergence of a MNIST latent space

Figure 6.12 shows the distribution of our data performed by the T-SNE algorithm. We used the saved models where the minimum MSE was achieved. We took the validation set and introduced it into the saved model. The output vectors from the encoder were used as the input of the T-SNE algorithm. We plotted the output points of the algorithm. Then we plotted 65 faces in the location of their corresponding point on the plot.

In Figures 6.12 (a) and (b) none pattern can be seen in the faces distribution . It seems this vectors doesnt represent any remarkable feature of the faces.

In Figures 6.12 (c), (d), (e) and (f) we drawed a red line. This red line separates the most part of the men from the most part of the women in each plot. It seems that ResNet and VGG structures are generating vectors that are able to represent somehow the gender of the subjects.

From this results we conlcude that VGG and ResNet, both trained with Incremental β training, are the best networks. We will use this decoders for testing the Neutral Face Generator from frontal faces. Both networks have been saved, VGG has a size of 1.8GB and ResNet has a size of 4.0GB.

6.5 Neutral Face Generator from frontal faces

6.5.1 Hyperparameters

After several test we concluded that the best hyperparameters for this networks were :

- Learning Rate: 1e-05
- Dropout Rate: 0.5
- Activation functions: ELU
- Latent dimension : 64
- Batch size : 16
- Optimizer : Adam

6.5.2 Neutral Face Generator from frontal faces networks results

In this section we will show the obtained results from the tests performed on VGG and ResNet. This networks were trained following the Incremental β training as it showed to give us good results on the Neutral Face Generator. Networks were trained in 4 different ways :

- From scratch.
- From scratch adding a regularization term.
- Attaching the pre-trained decoder with the weights freezed.

- Attaching the pre-trained decoder with the weights freezed and adding a regularization term.

As explained in Methodology the regularization term that we mentioned above is computed in the following way,

$$Regularization\ loss = |V_T - V_R| \tag{6.1}$$

were V_T represents the vectors obtanined by passing the neutral faces of the dataset we used on this part to the trained Neutral Face Generator trained previously. V_R represents the vectors that this new network is generating.

First we needed to decide how much frontal images we were going to use. For this we trained our best network (VGG with Incremental β training) from scratch with a different number of input images (3,5,7).

Network	Number of Images	Minimum MSE
VGG	3	52.03
VGG	5	46.54
VGG	7	44.88

Table 6.2: Minimum MSE performed by VGG network with different number of input images

As Table 6.2 shows results are worse using 3 input images and there is not very much difference between using 5 or 7 input images. In spite of this, using 7 input images performed better so we decided to use this number of input images for our next experiments.

Now we present a table showing : if we applied to the network the reguarization term, if we attached the pre-trained decoder, the minimum Mean Squared Error (MSE) achivied on the validation set, the epoch when its achived and the total epochs performed by each network.

Network	Regularization	Attached Decoder	Minimum MSE	Epoch of min MSE	Total Epochs
VGG	NO	NO	44.88	150	175
VGG	YES	NO	48.39	125	145
VGG	NO	YES	41.57	205	260
VGG	YES	YES	35.53	140	250
ResNet	NO	NO	53.8	50	60
ResNet	YES	NO	65.23	25	50
ResNet	NO	YES	35.67	165	485
ResNet	YES	YES	36.59	120	165

Table 6.3: Minimum MSE, Epoch were its achived and Total Epochs of each network with different training styles

On Table 6.3 we observe that using the regulatization term without the pre-trained decoder increases the minimum MSE. For both networks attaching the pre-trained decoder improved the performance deacreasing the minimum MSE.

When we train from scratch the VGG network the MSE seems to maintain good results. Adding the regularization term without the pre-trained decoder gets worse the network performance. Attaching the pre-trained decoder improved the results. Using the regularization term combined with the pre-trained decoder leads us to the best model.

For the ResNet network training from scratch doesn't seem to work. The network gets quicky overfitted and adding the regularization term gets things worse. In spite of this, attaching the pre-trained decoder changed the performance giving almost similar results as the best VGG model.

Now we will analyze the visual reconstruction performance of each network. Images can be checked in the next page in Figure 6.13. We will ignore the networks trained without the pre-trained decoder as it can be checked in Figures 6.13 (b), (c), (f) and (g) their performance isn't well compared with the networks trained with the pre-trained decoder, Figures (d), (e), (h) and (i).

In all the networks the first subject preserved the gender. VGG with pre-atrined decoder and no regularization term seems to recognise better this face geometry.

In all the networks the second subject preserved the gender. The networks had problems in recognizing this face geometry. VGG perfermed better, but none of the networks was able to reconstruct this face geometry correctly.

In all the networks the third subject preserved the gender. None of the networks was able to preserve ethnicity. VGG preserved better the make-up.

In all the networks the fourth subject preserved the gender. Only ResNet was able to preserve the beard. The 4 networks preserved well the face geomerty.

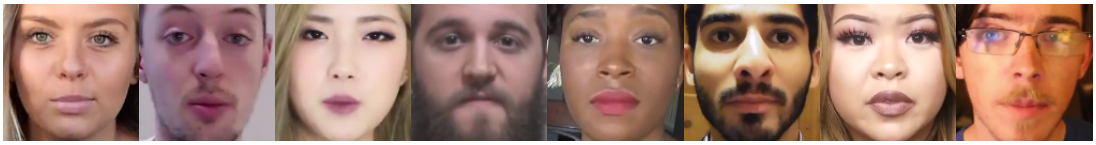
None network was able to preserve gender or ethnicity of the fifth subject.

In all the networks the sixth subject preserved the gender. Only ResNet was able to preserve the beard but it had troubles to reconstruct correctly the eyebrows.

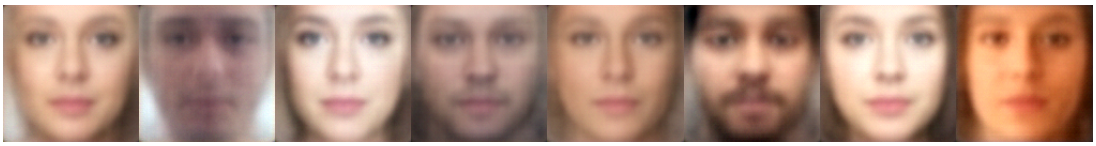
In all the networks the seventh subject preserved the gender. ResNet preserved better ethnicity and face geometry.

In all the networks the last subject preserved the gender. None network preserved glasses or beard. None of the networks was able to reconstruct this face geometry correctly.

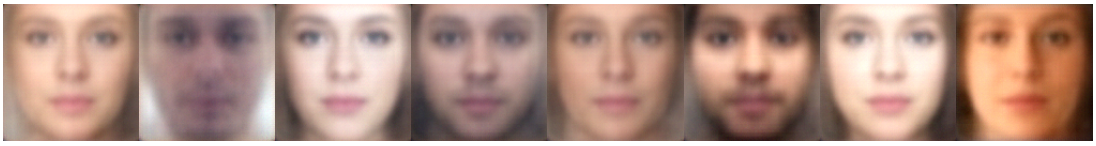
In general despite Table 6.3 shows that VGG has a better reconstruction performance than ResNet, seems that ResNet has a better visual reconstruction performance.



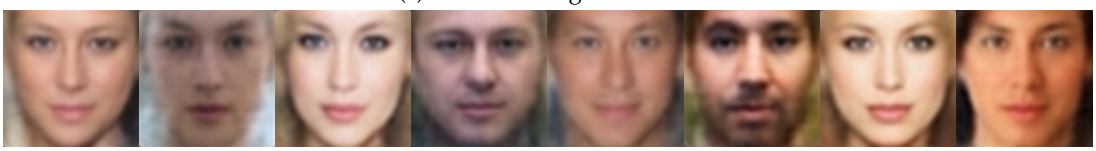
(a) Original Images



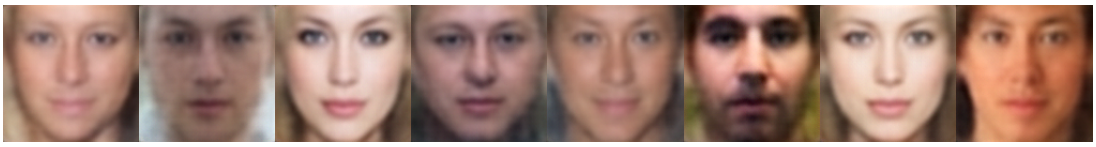
(b) VGG without regularization



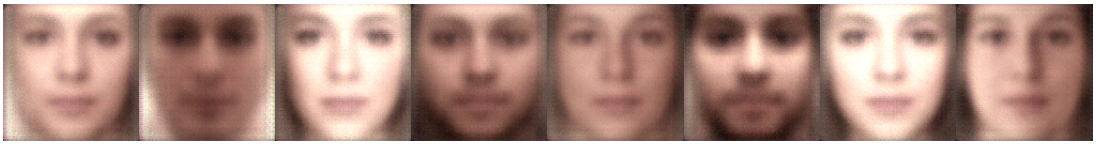
(c) VGG with regularization



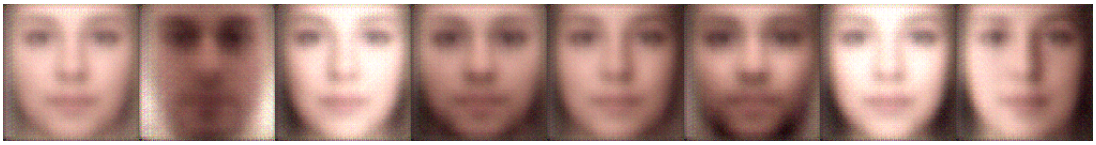
(d) VGG with freezed attached decoder and without regularization



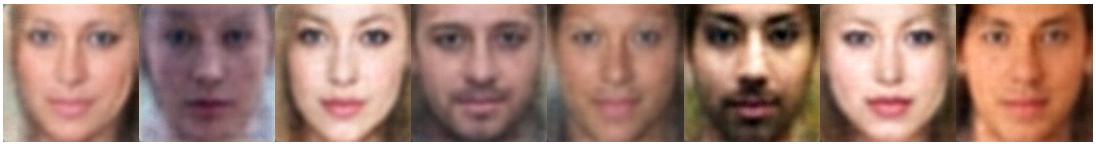
(e) VGG with freezed attached decoder and with regularization



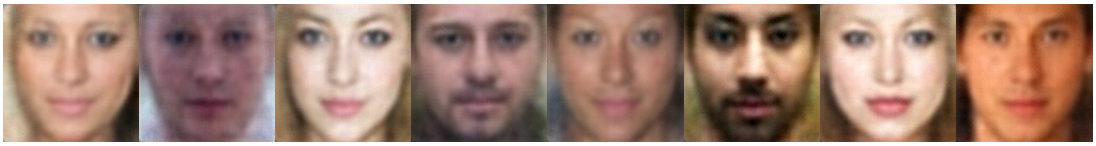
(f) ResNet without regularization



(g) ResNet with regularization



(h) ResNet with freezed attached decoder and without regularization



(i) ResNet with freezed attached decoder and with regularization

Figure 6.13: Reconstruction Images compared with the Originals of each of the networks. First row shows the original images, next rows show the recontruction performed by the networks.

Now we show 2 distribution plots from the best models. This models are VGG with pre-trained decoder plus regularization term and ResNet with pre-trained decoder.

(a) VGG with pre-trained decoder plus regularization term

(b) ResNet with pre-trained decoder

Figure 6.14: Distribution plots of the 2 best models.

In Figures 6.14 (a) and (b) we observe both models seem to generate vectors where the illumination and skin color is the main feature. We painted a red arrow to show the direction in which the network is organizing the faces with respect to ilumination and skins color. On the VGG plot (Figure 6.14 (a)) we painted a green circle indicating the location of the most part of the men, this network seem to preserve the gender recognition.

Chapter 7

Conclusions and Future work

In this section we will sum up all the results obtained in the project to give some concrete conclusions. And we will include some possible improvements for the future.

First, based on the results obtained on the Neutral Face Generator we can conclude that classical architectures as VGG and ResNet work well on Face Generation. Training Variational Autoencoders with incremental β training is a good way of maintaining a good reconstruction while preserving the generation power of the networks. Our best networks also showed that they are able to preserve gender and most part of the face geometry when performing reconstruction. Its also interesting how without performing an extensive study of the networks we have been able to generate a good diversity of features as gender, illumination or ethnicity.

Second, our idea of applying transfer learning worked well. In Figure 6.13 we observe that if we had trained the networks from scratch, the results would have been really poor. Adding the regularization term and the pre-trained decoder showed to improve the ability of the networks to generate neutral faces from the frontal ones. This idea could be applied in the future for generating other kind of expressions without needing big datasets.

The problem of generating a neutral face from frontal faces of the same subject continues open. But we have showed an approach were it isn't needed the use of very big datasets. Some possible improvements for this project are :

- Increase the size and quality of both datasets.
- Compare to GAN.
- Add different measures to classify generated images.
- Develop a program to generate neutral faces from input videos.
- Test deeper and different network structures.

Bibliography

[1] A. Korotcov, V. Tkachenko, D. P. Russo, and S. Ekins, "Comparison of deep learning with multiple machine learning methods and metrics using diverse drug discovery data sets," *Molecular pharmaceutics*, vol. 14, no. 12, pp. 4462–4475, 2017.

[2] A. Razavi, A. van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," in *Advances in Neural Information Processing Systems*, pp. 14866–14876, 2019.

[3] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[4] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8110–8119, 2020.

[5] D. Ravì, C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo, and G.-Z. Yang, "Deep learning for health informatics," *IEEE journal of biomedical and health informatics*, vol. 21, no. 1, pp. 4–21, 2016.

[6] H. Fujiyoshi, T. Hirakawa, and T. Yamashita, "Deep learning-based image recognition for autonomous driving," *IATSS research*, vol. 43, no. 4, pp. 244–252, 2019.

[7] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, 2008.

[8] D. O. Hebb, *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[9] S. Linnainmaa, "The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors," *Master's Thesis (in Finnish), Univ. Helsinki*, pp. 6–7, 1970.

[10] M. Alber, I. Bello, B. Zoph, P.-J. Kindermans, P. Ramachandran, and Q. Le, "Backprop evolution," *arXiv preprint arXiv:1808.02822*, 2018.

[11] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[13] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[14] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[15] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," tech. rep., Colorado Univ at Boulder Dept of Computer Science, 1986.

[16] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[17] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[18] H. Larochelle and Y. Bengio, "Classification using discriminative restricted boltzmann machines," in *Proceedings of the 25th international conference on Machine learning*, pp. 536–543, 2008.

[19] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011.

[20] G. E. Hinton and R. R. Salakhutdinov, "Replicated softmax: an undirected topic model," in *Advances in neural information processing systems*, pp. 1607–1614, 2009.

[21] M. A. Carreira-Perpinan and G. E. Hinton, "On contrastive divergence learning.," in *Aistats*, vol. 10, pp. 33–40, Citeseer, 2005.

[22] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, pp. 396–404, 1990.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[26] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

[27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[28] S. J. Thorpe and M. Fabre-Thorpe, "Seeking categories in the brain," *Science*, vol. 291, no. 5502, pp. 260–263, 2001.

[29] K. Motohashi, H. Takagi, S. Maruyama, S. Ookawara, T. Takeuchi, H. Koike, S. Hanaoka, K. Hirose, H. Cleaves, and H. Funakubo, "Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake.," *Physical Review Letters*, vol. 114, no. 19, p. 191803, 2015.

[30] K. Hornik, M. Stinchcombe, H. White, *et al.*, "Multilayer feedforward networks are universal approximators.," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[31] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[33] M. Alkhayrat, M. Aljnidi, and K. Aljoumaa, "A comparative dimensionality reduction study in telecom customer segmentation using deep learning and pca," *Journal of Big Data*, vol. 7, no. 1, p. 9, 2020.

[34] P. Vincent, "A connection between score matching and denoising autoencoders," *Neural computation*, vol. 23, no. 7, pp. 1661–1674, 2011.

[35] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[36] S. Odaibo, "Tutorial: Deriving the standard variational autoencoder (vae) loss function," *arXiv preprint arXiv:1907.08956*, 2019.

[37] E. Friesen and P. Ekman, "Facial action coding system: a technique for the measurement of facial movement," *Palo Alto*, vol. 3, 1978.

[38] S. Villagrasa and A. Susín Sánchez, "Face! 3d facial animation system based on facs," in *IV Iberoamerican symposium in computer graphics*, pp. 203–209, 2009.

[39] S. K. D'mello and A. Graesser, "Multimodal semi-automated affect detection from conversational cues, gross body language, and facial features," *User Modeling and User-Adapted Interaction*, vol. 20, no. 2, pp. 147–187, 2010.

[40] K. L. Schmidt and J. F. Cohn, "Dynamics of facial expression: Normative characteristics and individual differences," in *IEEE International Conference on Multimedia and Expo, 2001. ICME 2001.*, pp. 140–140, Citeseer, 2001.

[41] L. I. Reed, M. A. Sayette, and J. F. Cohn, "Impact of depression on response to comedy: A dynamic facial coding analysis.," *Journal of abnormal psychology*, vol. 116, no. 4, p. 804, 2007.

[42] A. C. Lints-Martindale, T. Hadjistavropoulos, B. Barber, and S. J. Gibson, "A psychophysical investigation of the facial action coding system as an index of pain variability among older adults with and without alzheimer's disease," *Pain Medicine*, vol. 8, no. 8, pp. 678–689, 2007.

[43] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264, IGI global, 2010.

[44] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[45] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database forstudying face recognition in unconstrained environments," 2008.

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[47] Z. Li, J. V. Murkute, P. K. Gyawali, and L. Wang, "Progressive learning and disentanglement of hierarchical representations," *arXiv preprint arXiv:2002.10549*, 2020.

[48] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," 2016.

[49] T. Baltrusaitis, A. Zadeh, Y. C. Lim, and L.-P. Morency, "Openface 2.0: Facial behavior analysis toolkit," in *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pp. 59–66, IEEE, 2018.

[50] A. Zadeh, Y. Chong Lim, T. Baltrusaitis, and L.-P. Morency, "Convolutional experts constrained local model for 3d facial landmark detection," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 2519–2528, 2017.

[51] T. Baltrusaitis, P. Robinson, and L.-P. Morency, "Constrained local neural fields for robust facial landmark detection in the wild," in *Proceedings of the IEEE international conference on computer vision workshops*, pp. 354–361, 2013.

[52] T. Baltrušaitis, M. Mahmoud, and P. Robinson, "Cross-dataset learning and person-specific normalisation for automatic action unit detection," in *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, vol. 6, pp. 1–6, IEEE, 2015.

[53] E. Wood, T. Baltrusaitis, X. Zhang, Y. Sugano, P. Robinson, and A. Bulling, "Rendering of eyes for eye-shape registration and gaze estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3756–3764, 2015.

[54] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[55] H. J. Escalante, H. Kaya, A. A. Salah, S. Escalera, Y. Gucluturk, U. Guclu, X. Baró, I. Guyon, J. J. Junior, M. Madadi, *et al.*, "Explaining first impressions: Modeling, recognizing, and explaining apparent personality from videos," *arXiv preprint arXiv:1802.00745*, 2018.